Títol:	Anàlisi i integració de models de programació paral·lels en SoC Tegra 2
Volum:	1/1
Alumne:	David Prat Robles
Director/Ponent:	Alejandro Ramírez Bellido
Departament:	Arquitectura de Computadors
Data:	19 de Gener de 2011

DADES DEL PROJECTE

Títol del Projecte: Anàlisi i integració de models de programació paral·lels en SoC Tegra 2

Nom de l'estudiant: David Prat Robles
Titulació: Enginyeria Informàtica
Crèdits: 37,5
Director/Ponent: Alejandro Ramírez Bellido
Departament: AC

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Jesús José Labarta Mancho

Vocal: Conrado Martínez Parra

Secretari: Alejandro Ramírez Bellido

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

Anàlisi i integració de models de programació paral·lels en SoC Tegra 2

David Prat Robles

19 de Gener de 2011

Agraïments

Aquest projecte que he dut a terme té al darrera persones que m'han ajudat abans i durant la seva realització. M'agradaria dedicar-lo per tant a totes elles.

Al Jesús Labarta per haver-me ofert el projecte.

A l'Àlex Ramirez per haver-me ensenyat com presentar l'estudi sobre la placa de desenvolupament i per haver estat tant atent amb mi durant la realització del projecte.

A l'Àlex Duran, per haver-me ajudat en la part de la integració de NANOS++.

Al David Chait, per haver-me ajudat amb alguns problemes amb la placa de desenvolupament.

A tots els professors que he tingut a la facultat, especialment els dels últims anys, per la seva professionalitat.

A la meva família i amics per donar-me suport i ànims durant tot aquest temps.

I en general, a tothom que m'ha donat un cop de mà per aconseguir que aquesta aventura hagi estat un èxit.

Per últim, m'agradaria afegir unes paraules que si bé no pretenen ser cap veritat, si que tenen la intenció de ser una interpretació de la realitat que pugui ajudar a qui les llegeixi.

"Tot el que no s'expandeix acaba morint, el que s'expandeix també.
La transcendència de la totalitat està en el progrés que provoquen les expansions."

David Prat

Índex

Capítol 1

1 Introducció

1.1 Motivació del projecte

Després d'assolir els perfils ACAR, (Arquitectures i computació d'alt rendiment) i XTSO, (Xarxes telemàtiques i sistemes operatius), de la FIB, vaig decidir que volia fer un projecte de final de carrera que englobes el màxim possible de disciplines en que m'havia especialitzat amb l'objectiu de poder aplicar tot el que havia après.

Gràcies a haver-me especialitzat en aquestes branques i voler fer un projecte que les tractes, he pogut desenvolupar aquest projecte, "Anàlisi i integració de models de programació paral·lels en SoC Tegra 2", tenint una visió global de tot el conjunt i entrant en el màxim detall possible en els factors realment importants.

1.2 Estat de l'art

1.2.1 Una mica d'història

Els orígens dels supercomputadors van lligats a Seymour Cray, nord-americà nascut el 1925. Després d'una trajectòria que va passar des de ser un dels dissenyadors de l'UNIVAC 1103 el 1953, passant per crear el CDC 6600 el 1963, que va superar la màquina més potent de IBM, (a l'empresa CDC, de la qual ell va ser un dels fundadors junt amb altres enginyers el 1957); va fundar l'empresa Cray Research el 1972, on va crear el que seria el primer supercomputador més representatiu de la història, el Cray-1.

El Cray-1 va ser posat en funcionament al Laboratori nacional de Los Alamos al 1976. Era capaç d'operar a 250 MFLOPS. Consumia 115 KW sense comptar el sistema de refrigeració, que s'estima que doblaria el consum total. Per poder arribar a computar amb tal capacitat va fer ús per primer cop del processador vectorial. No tenia jerarquia de memòria però, tenia prous bancs de memòria com per encadenar peticions a diferents bancs augmentant així l'ample de banda de memòria al processador.

Figura 1.1: Cray-1

El 1985, Cray Research, es va reafirmar com a líder del mercat amb el Cray-2/8, que operava a 3,9 GFLOPS; va assolir una quota del mercat del 70%. El 1989 el Cray-2 va ser superat per el ETA10-G/8 que era capaç d'operar a 10,3 GFLOPS; ETA10-G/8 va ser creat per una divisió spin-off de CDC.

A partir del 1990 Japó va entrar en el mercat dels supercomputadors amb el NEC SX-3/44R, que va ocupar la primera posició mundial amb 23,2 GFLOPS.

El 1993 Intel va ocupar la primera posició amb el Intel Paragon XP/S 140 amb 143,4 GFLOPS.

El 2002 Japó va tornar a ocupar la primera posició amb el NEC Earth Simulator que operava a 35,9 TFLOPS. Va ser un cop fort per als supercomputadors nord-americans, ja que l'anterior supercomputador més potent havia estat el IBM ASCI White amb 7,2 TFLOPS. Va ser una diferència de còmput important.

Com a contrapartida, IBM va crear el Blue-gene L; va ocupar la primera posició des del 2004 fins al 2007, arribant als 478,2 TFLOPS. Al 2008 es va crear el IBM Roadrunner amb 1,1 PFLOPS i el 2009 el Cray Jaguar amb 1,7 PFLOPS, [18], cal a dir que l'empresa que va crear el Cray Jaguar, Cray Research va ser comprada per SGI el 1996 després de que quedes en fallida i tornada a comprar el 2000 per Tera Computer Company, [22*].

Figura 1.2: Cray Jaguar XT5-HE

Des del Cray-1 al 1976, en poc menys de 30 anys la velocitat de computació dels supercomputadors actuals és de més d'un milió de vegades major. Però, el seu consum i la seva refrigeració s'estan convertint en un dels problemes més importants per a poder seguir escalant la potència de còmput, és per això que és fonamental que les tecnologies implicades segueixin evolucionant.

1.2.2 La supercomputació a Espanya

La supercomputació a Espanya gira entorn al Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). BSC-CNS és un consorci públic entre el Ministerio de Educación y Ciencia espanyol, la Generalitat de Catalunya i la Universitat Politècnica de Catalunya (UPC) constituït oficialment a l'abril del 2005, i dirigit pel doctor Mateo Valero, també professor de la UPC. Un dels objectius principals del centre és donar servei a la investigació oferint recursos de supercomputació per al progrés científic.
El BSC–CNS té com a principal eina el supercomputador MareNostrum. Es va posar en funcionament el 12 d'abril de 2005 presentat per l'empresa IBM i per la ministra española d'educació i ciencia María Jesús San Segundo. El Novembre de 2006 va arribar ocpuar la cinquena posició del Top500, la llista dels supercomputadors més potents del món.

Situat al rectorat l'antigua capella de la Torre Girona de 1920; està considerat el supercomputador més bonic del món.

Figura 1.3: MareNostrum

A continuació veiem les especificacions més rellevatns del MareNostrum.

Nodes

- 2560 JS21, 2.3GHz.
- 4 cores per placa.
- 8 Gbytes.
- 36 Gbytes disc SAS.
- 2560 JS21, 2.3GHz.

Xarxa

- Myrinet.
- 2 Spine 128036.
- 10 Clos256.
- 2560 Targes Myrinet.
- Gigabit.

Disc

- Sobre 40 servidors de disc.

Pel que fa les communicacions per computar, els nodes del MareNostrum estan interconnectats mitjançant una xarxa Myrinet amb topologia fat tree. Amb 256 links (1 a cada node), 250MB/s de bandwith a cada direcció. On sabem que si som capaços d'executar l'aplicació en un mateix clos tindrem més rendiment.

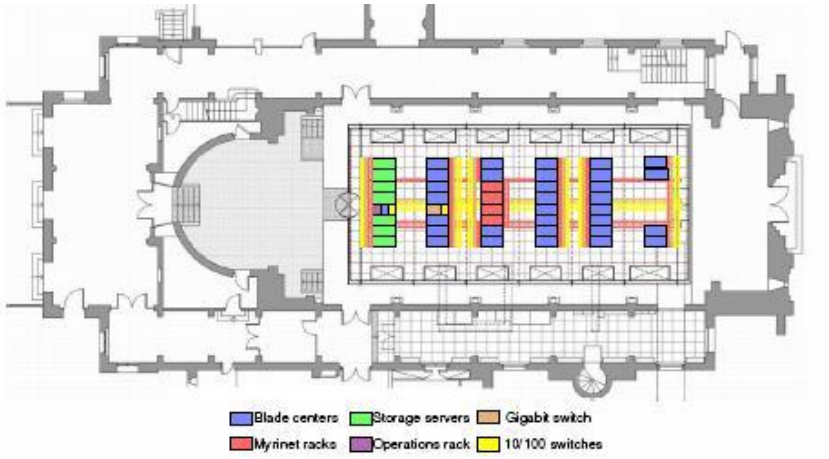

Figura 1.4: Funcionalitat dels racks del MareNostrum

Fruit de la necessitat de fer més potent MareNostrum, es van aprofitar els antics blades per formar una xarxa de supercomputació a Espanya, (RES). Vegem la situació al 2009.

Figura 1.5: Topologia de la xarxa RES a Espanya

Supercomputador/clúster	Emplaçament	Nom del centre	TFLOPS
MareNostrum	Universitat Politècnica de Catalunya	BSC-CNS	94.208
Magerit	Universidad Politecnica de Madrid	UPM	21.190
LaPalma	Instituto de Astrofísica de Canarias	IAC	4.506
Altamira	Universidad de Cantabria	UC	4.506
Picasso	Universidad de Málaga	UM	4.506
Tirant	Universidad de Valencia	UV	4.506
CaesarAugusta	Universidad de Zaragoza	UZ	4.506

Taula 1.1: Supercomputadors dels centres de la RES

1.2.3 En l'actualitat

Actualment, la mitja de rendiment dels 500 supercomputadors més potents del món segons la llista de Top500 és de 24,67 TFLOPS, (llista del Top500 de Juny de 2010). El primer de tots arriba a 1759 TeraFLOPS, (1,76 PetaFLOPS). Si mirem la llista Top500, (www.top500.org), veurem que entre els 15 supercomputadors més potents i la resta hi ha molta diferència de rendiment.

Figura 1.6: Rendiment dels 500 supercomputadors més potents del món al llarg del temps

És a dir, que podem distingir entre els anomenats "hero supercomputers", que porten al límit l'escalabilitat de nodes amb les tecnologies actuals, (els 5 primers de la llista), supercomputadors molt potents que van des de 700 TFLOPS fins a un mínim de 200 TFLOPS, (del 6 al 22 de la llista), i la resta de la llista.

1.2.3.1 Xips actuals

En quan a nuclis de processament, a grans trets, existeixen 4 estratègies per assolir FLOPS.

Per una banda hi ha el que són els processadors clàssics que s'usen per a servidors de CPDs, on Intel i AMD són els principals venedors. Són els que entren dins del que s'anomena "comodity hardware".

Més especialitzats serien els PowerPC de IBM dissenyats amb l'objectiu de la computació d'alt rendiment. Tenen com a filosofia consumir pocs Watts per així poder tenir més processadors en total.

Finalment distingim entre dues alternatives que actualment encara són poc usades. Per una banda tenim processadors heterogenis com ara el Cell, que disposen d'acceleradors dins del mateix chip, en el que s'anomena un network on chip. Per l'altre banda tenim les targes gràfiques en la seva versió GPGPU. Aquestes dues últimes opcions són més difícils de programar però, a canvi ofereixen molts més FLOPS per Watt consumit, que com anirem veient durant tota la memòria, aquest paràmetre es convertirà en la pedra angular a l'hora de dissenyar els supercomputadors del futur.

Processador	FLOPS	Consum en Watts	MFLOPS/W
IBM PowerXCell 8i 3200 MHz	(12,8 GFlops 1x spu) 102,4 GFLOPS	92	1113,1
NVIDIA Tesla C2050 GPU, [18*]	515,2 GFLOPS	247	2085,83
IBM Power 780	(33,1 1x core) 264,96 GFLOPS	X	X
AMD opteron Six Core 2600 MHz, [16*]	10,4 GFLOPS	75	138,7
Intel EM64T Xeon E54xx (Harpertown) 3000 MHz, [17*]	12 GFLOPS	80	150
PowerPC 450 850 MHz	3,4 GFLOPS	X	X

Taula 1.2: Rendiment de diferents tipus de tipus d'arquitectura

Com veiem la GPU rendeix 14 vegades més per Watt que el processador de Intel, no obstant les GPUs van ser creades per processament de gràfics i necessiten una programació considerablement més elaborada.

Vegem ara d'entre la llista del 500 supercomputadors més potents quines famílies de processadors han estat les més usades al llarg del temps fins al juny de 2010.

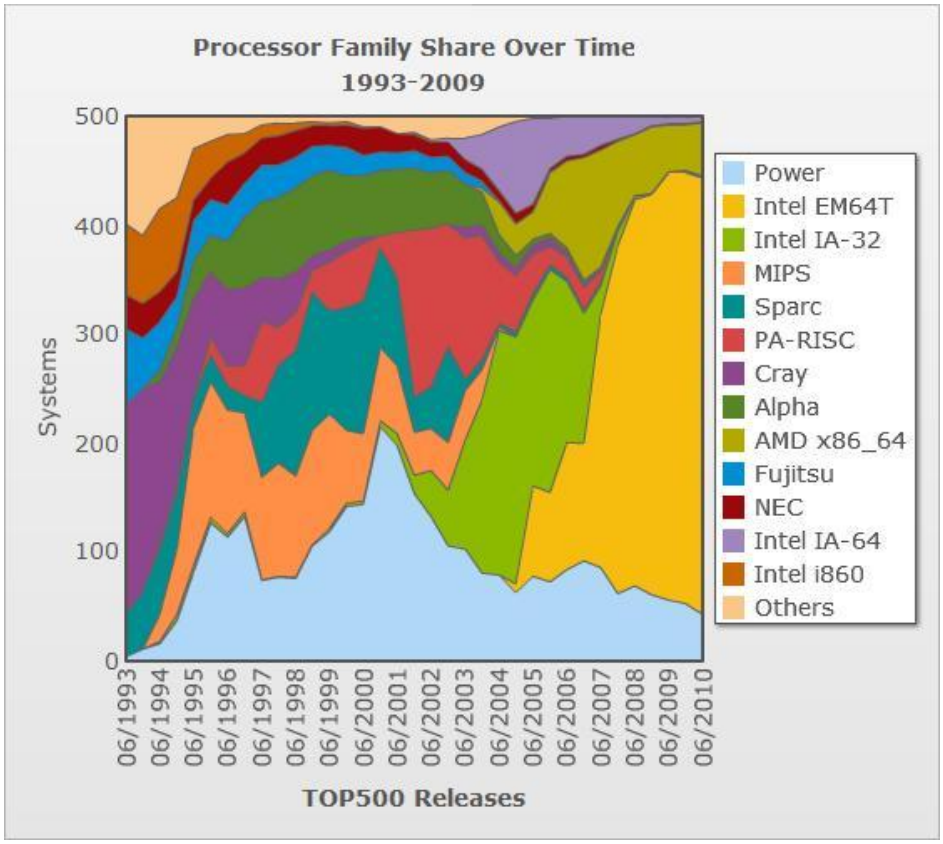

Figura 1.7: Famílies de processadors usades pels 500 supercomputadors més potents del món fins al Juny de 2010

Es pot apreciar com els més usats són la família dels els Intel EM64T, els segueixen els Power de IBM i la competència més directa, els AMD x86_64. El que passa és que és molt car tornar a compilar tots els programes per una altra arquitectura i la x86 va dominar en els últims 6 anys, després de que Alpha desapareixes.

1.2.3.2 Dades d'alguns supercomputadors actuals

En aquesta secció es mostra quins són els supercomputadors més potents actualment, (llista del Top500 i Green500 de Juny de 2010), quina família de xip usen i com influeix aquesta en els paràmetres més significatius d'un supercomputador, [14*], [15*].

	FLOPS	MW	MFLOPS/W	Nuclis	Posició Green500	Posició Top500	Any	Tipus
Cray XT5-HE Opteron Six Core 2.6 GHz	1,75 PFLOPS	6,95 MW	235,77	224162	64	1	2009	Processadors servidors
Dawning Nebulae, TC3600 blade CB60-G2 cluster, Intel Xeon 5650/ nVidia C2050, Infiniband	1,27 PFLOPS	2,58 MW	492.64	120640	4	2	2010	Híbrid processadors servidors amb GPUs
Roadrunner (BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband), [20]	1,04 PFLOPS	2,35 MW	444.25	122400	7	3	2008	Híbrid processadors servidors amb Processador amb SPUs
JUGENE - Blue Gene/P Solution 2009	825.50 TFLOPS	2,26 MW	363.98	294912	19	5	2009	Processadors baix consum
Marenostrum (MN)	63,83 TFLOPS	1,2MW	53,19	10240	X	87	2005	Processadors servidors
Cray-1	80 MFLOPS	0,115 MW	0,000695	X	X	X	1973	X
GRAPE-DR accelerator Cluster, Infiniband	21,96 TFLOPS	28.67 KW	815.43	8192	1	445	2010	Híbrid Intel Core i7-920 + x58 chipset

Taula 1.3: Especificacions de diferents tipus de supercomputadors

Veiem que entre els 5 primers supercomputadors més ràpids del món hi han tots els tipus de nuclis que es poden fer servir. El més potent, el Jaguar XT5-HE només usa processadors AMD de 6 nuclis; el Dawning Nebulae usa una combinació de Intel Xeon 5650 amb nVidia tesla C2050; el Roadrunner també usa una combinació però, de AMDs de dos nuclis i Cells; el Jugene usa només PowerPC 450.

Cadascun dels paradigmes té els seus pros i contres en relació als demés.

	Jaguar	**Dawning Nebulae**	**Roadrunner**	**Blue Gene/P Jugene**
Paradigma	Multicore	Híbrid, CPUs+GPUs	Híbrid, CPUs+Acceleradors	Manycore, cores grossos.
Consum	Molt alt	Baix	Baix	Alt
MFlops/Watt	Baix	Alt	Alt	Mitjà
Cores	Molts	Bastants	Bastants	Molts, el que en té més, el seu paradigma es basa en tenir-ne molts.
Dificultat de programació	Mitjana, MPI amb OpenMP	Alta; s'han de programar els codis per fer treballar les GPUs, Nvidia dona suport amb CUDA	Molt alta. MPI amb OpenMP, a demés de programar PPEs i SPEs del Cell.	Mitjana, MPI amb OpenMP

Taula 1.4: Característiques dels supercomputadors més potents del món

Com veiem el repte actual és aconseguir millorar al màxim els MFLOPs/Watt ja que per arribar al PFLOP els consums actuals són prohibitius per la majoria de centres de càlcul. És més, tenint en compte que actualment cada Watt de potència consumida per còmput requereix 1 Watt de potència per refrigerar, el Cray XT5-HE necessita en total 13,9 MW de potència per funcionar.

La propera barrera és el ExaFLOP per tant, fent una conversió directa, un sistema com el Cray XT5-HE necessitaria 7942 MW per poder arribar a 1 ExaFLOP. Sabent que la potència mitjana dels reactors nuclears és de 1000 MW. Una central nuclear amb un reactor són entre 1500 i 2000 milions de dòlars, es triga 5 anys en construir, (a la pràctica pot arribar a ser 3300 milions de dòlars i 8 anys) i necessita entre 1 i 4 kilòmetres quadrats. Ens queda que necessitaríem 8 centrals nuclears per poder alimentar el sistema, cosa que és inviable.

Una altre factor a tenir en compte que és el nombre de cores s'acaba traduint en el nombre de nodes que té el sistema complet. Per tenir una xarxa d'interconnexió amb una latència baixa un nombre reduït de nodes és un factor clau. Per exemple, veiem com el Cray XT5-HE i el Jugene no tenen assequibles una xarxa fat tree sinó que han d'usar una torus3D que té més latència, (encara que aquesta escala millor).

En el nombre de nodes també queda inclòs el temps mig de fallada dels components, contra més processadors i nodes tinguem, més peces hardware i per tant menys temps entre fallades hardware. Per exemple, el Cray XT5-HE pot estar executant un programa disposant de tots els processdors durant unes 32 hores de mitjana.

D'altre banda programar en paral·lel per aquests monstres és tot un repte, només cal saber que els IDCs venen aplicacions que en la seva gran majoria només escalen fins a 32 nuclis. Si ens fixem en el sistemes heterogenis aquests s'han de programar expressament i els codis no són reutilitzables. És a dir, es necessita gent molt qualificada, amb tot el que això implica.

1.2.4 Qui necessita tants FLOPS?

Sabent que és tan difícil fer funcionar un supercomputador i que la seva penetració social és baixa. Ens podem preguntar; qui necessita tants FLOPS?

El 2006, Steven Chen, un enginyer que va treballar molt temps amb Seymour Cray i va treballar dissenyant el Cray X-MP, va fer una conferència on es va parlar d'algunes de les aplicacions i quin potencial de càlcul necessitaven, [11], [26*]:

- Desenvolupament per automoció: 100 TeraFLOPS.
- Simulació de visió humana: 100 TeraFLOPS.
- Anàlisi de aerodinàmica: 1 PetaFLOPS.
- Òptica amb laser: 10 PetaFLOPS.
- Dinàmica molecular en biologia: 20 PetaFLOPS.
- Disseny aerodinàmic: 1 ExaFLOPS.
- Cosmologia computacional: 10 ExaFLOPS.
- Turbulència en la física: 100 ExaFLOPS.
- Química computacional: 1 ZettaFLOPS.

Ens trobem amb que hi ha aplicacions que van fins al ExaFLOP i el ZettaFLOP, que són 1000 vegades més i 1000.000 vegades més que un PetaFLOP. És més, si tenim en compte que volem córrer varies instàncies per exemple d'aplicacions que són de 1 PetaFLOP, ens pot resultar fàcilment necessitar 20 PetaFLOPS.

Hom es pot preguntar si no és més barat fer els experiments que les simulacions al computador. Sol ser més car fer els experiments reals, per exemple llogar un túnel de vent és molt car si es lloga durant moltes hores i normalment el que un vol és fer molts experiments fins a tenir un bon disseny. Amb la simulació física fer molts experiments no és assequible econòmicament.

També hi ha experiments que amb les eines tecnològiques actuals no es poden dur a terme i s'han de fer amb simulació. Per exemple; mesurar totes les molècules d'aire que passen per una ala d'un avió, mesurar el procés d'un reactor nuclear, mesurar efectes

d'un medicament en el cos humà, etc. Una altre de les aplicacions, aquesta més polèmica, és esbrinar quin és el comportament del arsenal nuclear al pas del temps per saber si aquest representa un perill.

1.2.5 Els supercomputadors del futur

1.2.5.1 Problemàtica

Com hem vist, el principal obstacle a salvar és el consum d'electricitat; també hem vist que els sistemes híbrids ens poden estalviar molta energia ja que fan servir GPUs i/o processadors heterogenis que tenen més rendiment per Watt que els processadors clàssics no obstant, tenen com a principal desaventatge que són més difícils de programar.

Fem una mica d'aritmètica per veure quan rendiment per Watt haurien de poder tenir perquè la majoria de centres de càlcul poguessin tenir un PetaFLOP de còmput de manera assequible, [19*].

Agafem la mitjana del consum en Watts de totes les potències consumides declarades del Top500 i la mitjana del potencial de càlcul dels sistemes que han declarat la seva potència, sent aquestes 398 KW i 72 TFOPS respectivament, (llista de Juny de 2010), amb el que resulta que el potencial de còmput per Watt del mitjà és de 183 MFLOPS/Watt. Agafem el potencial de còmput per Watt del sistema híbrid més potent actual, el Dawning Nebulae, 492.64 MFLOPS/W que rendeix 2,7 vegades més que la mitja .

492,64 MFLOPS/W x 398 KW x 1000 W / 1 KW x 1 PFLOP /1.000.000.000 MFLOPs = 0,196 PFLOPS

Resulta que amb el que paguen de mitjà actualment de consum elèctric amb un sistema híbrid de processadors comodity hardware i GPUs estaríem molt lluny del PetaFLOP.

Si agafem el rendiment del sistema més eficient de la Little Green List, el GRAPE-DR accelerator Cluster, Infiniband. Amb un rendiment de 815.43 MFLOPS/W.

815.43 MFLOPS/W x 398 KW x 1000 W / 1 KW x 1 PetaFLOPs /1.000.000.000 MFLOPS = 0,325 PetaFLOPS

Encara estem lluny del PetaFLOP. Ara fem el contrari i busquem quants MPFLOPS/W hauria de tenir el sistema per poder-lo alimentar amb 398 KW.

1 PFLOPS x (1000.000.000 MFLOPS / 1 PFLOPS) / (398 KW x 1000 W / 1 KW) = 2512,56 MFLOPS/W.

És a dir que les tecnologies implicades han de permetre un rendiment 5,1 vegades major en els sistemes per tal de poder computar amb 1 PFLOP consumint 398 KW; Que és el consum de la mitjana dels sistemes del Top500 que han reportat el seu consum.

1.2.5.2 Predicció sobre sistemes futurs

El rendiment màxim, (pic), d'un processador a grans trets es pot calcular com:

FLOPS (pic) = freqüència * nombre d'operacions per cicle de rellotge * nombre de nuclis.

Anem a veure que ens ofereixen Intel AMD i NVIDIA pel 2011 i quines característiques poden arribar a tenir els seus productes al 2013. D'aquesta manera podem fer una predicció aproximada de quants nodes hauran de tenir els supercomputadors per arribar a un PetaFLOP de rendiment pic.

	2011		2013	
	Intel Sandy Bridge	AMD Interlagos	Intel Haswell	AMD ?
Freqüència en GHZ	3,4	2,66	3,8	3
FLOPS per cicle	8	8	8	8
Nuclis (threads)	4/6/8 (8/12/16)	6-16	18	24
Sockets per node	2	4	2	4
GFLOPS de pic per node	435,2	1361,92	1094,4	2304
Nodes per assolir el PetaFLOP de pic	2297,7	734,26	913,24	434,02
Eficiència de la xarxa	0,85	0,85	0,85	0,85
Nodes reals per assolir el PetaFLOP de pic	2704	865	1076	512

Taula 1.5: Predicció del nombre de nodes amb processadors comodity hardware

Ara fem la mateixa taula suposant que els nodes siguin híbrids amb CPUs comodity hardware i GPUs.

	2011		2013	
	Intel Sandy Bridge + GPU1	AMD Interlagos + GPU1	Intel Haswell + GPU2	AMD ? + GPU2
Freqüència en GHZ	3,4	2,66	3,8	3
FLOPS per cicle	8	8	8	8
Nuclis (threads)	4/6/8 (8/12/16)	6-16	18	24
Sockets per node proc	2	2	2	2
GFLOPS proc	435,2	680,96	1094,4	1152
Sockets per node GPU	2	2	2	2
GFLOPS GPU	1030,4	1030,4	2000	2000
Nodes per assolir el PetaFLOP de pic	683	418	324	318
Eficiència de la xarxa	0,85	0,85	0,85	0,85
Nodes reals per assolir el PetaFLOP de pic	804	492	382	375

Taula 1.6: Predicció nombre de nodes amb processadors amb GPUs

Amb els processadors projectats pel 2011, un 35,13% dels FLOPS es poden assolir amb els processadors, el 64,87% restant es correspon amb el que donarien les GPUs. Per al 2013 el percentatge pels processadors no augmenta gaire si contem amb que NVIDIA i ATI, (AMD), desenvoluparan GPUs amb el doble de FLOPS. Un 35,96% mentre que les GPUs donen el 64,03% del total. Les xifres són molt bones pel que fa a assolir el PetaFLOP amb pocs nodes no obstant, degut al baix rendiment que s'està traient actualment a les GPUs seria convenient preguntar-se si desenvolupar una tecnologia que permetés posar més sockets per node sense perdre rendiment no seria una altre opció com a alternativa a l'ús de GPUs als sistemes híbrids.

1.2.5.3 Tendència d'escalabilitat i alguns sistemes anunciats

En el lloc web de Top500 podem veure una gràfica sobre l'evolució de la potència de còmput al llarg del temps. A partir de les dades del passat i actuals sobre la potència de còmput del sistema més potent, l'últim, i la suma dels 500 es pot fer la tendència de les 3 potencies i projectar la potencia de càlcul que tindrem més enllà del 2010.

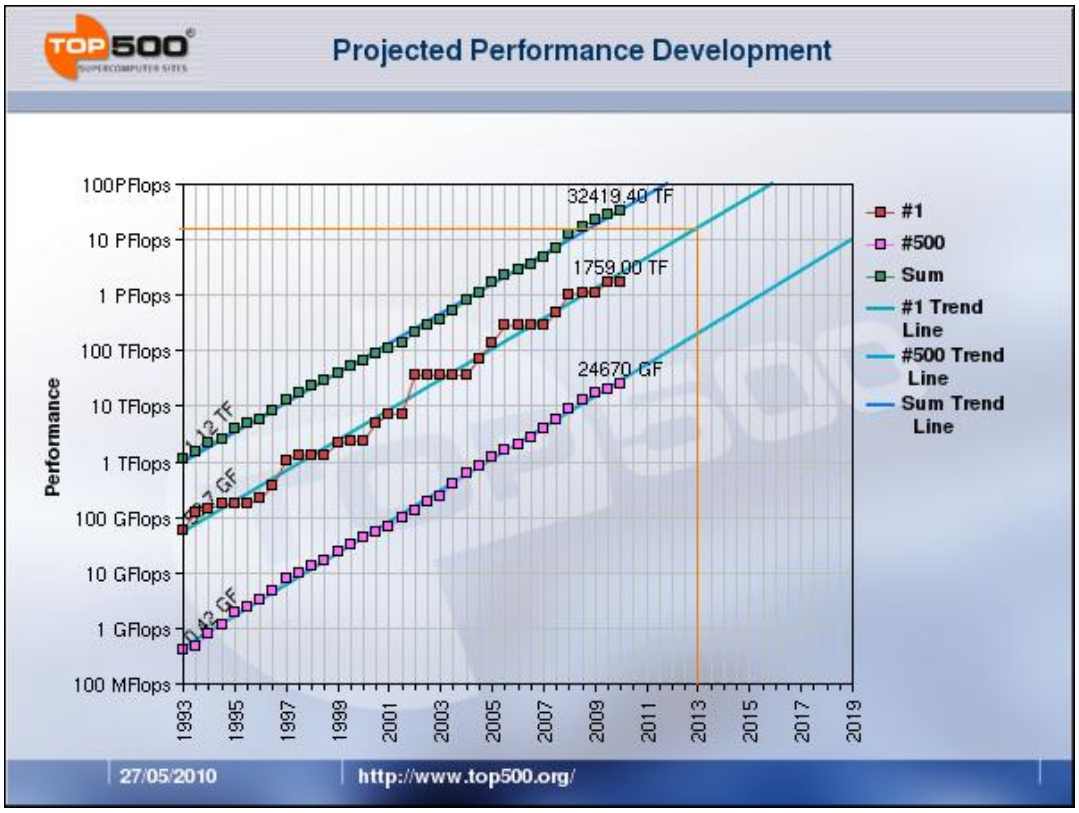

Figura 1.8: Rendiment projectat per als 500 supercomputadors més potents del món

Si ens fixem en la línia taronja, ens surten projectats uns 24 PetaFLOPS per al 2013 per al sistema més potent, cosa que sembla poc tenint en compte els rendiments que teòricament es podran assolir el 2013 amb els xips anunciats per diversos fabricants i els sistemes anunciats per abans del 2013. El que ens pot portar a creure-ho pot ser que les memòries DRAM no evolucionen tant ràpid com els processadors o que les xarxes d'interconnexió no estiguin preparades per tals amples de banda.

Anem a veure una gràfica amb més anys per poder comprovar si al llarg del temps hi ha hagut algun avenç més gran que el que hi ha projectat per 2013 segons el Top500.

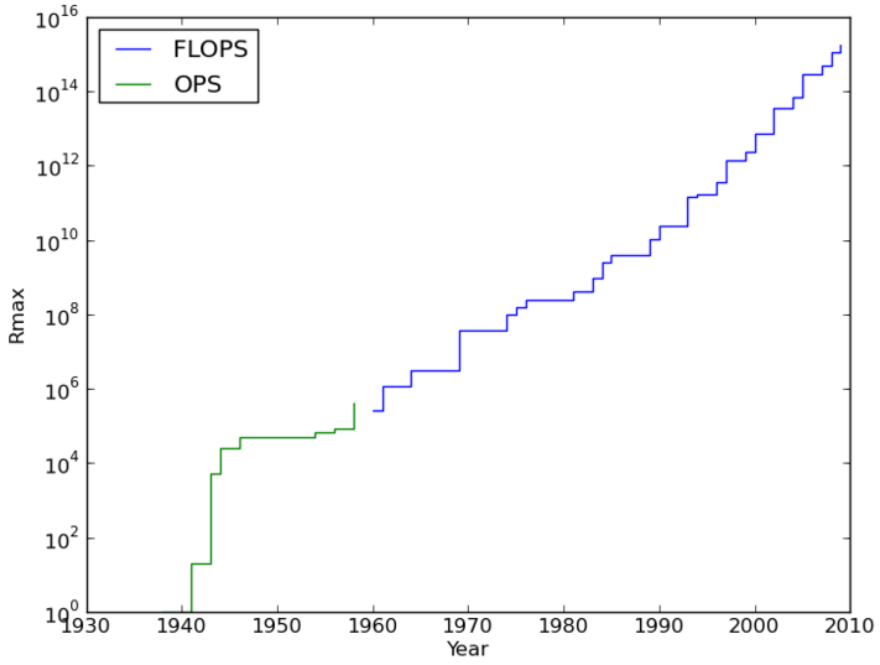

Figura 1.9: Velocitat a escala logarítmica vs. Temps

En efecte veiem que al llarg del temps s'han produït salts de rendiment molt més pronunciats, com per exemple entre els anys 60 i 70 i posteriorment als 90.

Vegem ara alguns supercomputadors anunciats per al futur, [32*], [33*]:

- ORNL, 2 PFLOPS (Jaguar), Cray (Opteron/Magny-Cours), 2009.
- RIKEN, 10 PFLOPS supercomputer, Fujitsu (Sparc64 VIIIfx), 2010.
- NCSA, 10 PFLOPS (Blue Waters), IBM (POWER), 2010.
- NASA, 10 PFLOPS (Pleiades), SGI (Altix ICE/Xeon), 2010.
- DOE NNSA, 20 PFLOPS (Sequoia), IBM (Blue Gene), 2011. Tindrà un nombre similiar de nodes pero amb més nuclis. Consumirà uns 6MW.

Si per al 2011 ja tenim 20 PFLOPS i els fabricants anuncien processasdors per al 2013 amb 100 vegades més rendiment en FLOPS; queda demostrat doncs que per al 2013, tal com s'aproxima en la secció de la memòria "Predicció sobre sistemes futurs", el més probable és que tinguem més rendiment que 24 PFLOPS com està projectat al Top500.

1.3 Objectius del projecte

Aquest projecte té com a principal objectiu fer ús del SoC Tegra 250 per obtenir el major rendiment de còmput des d'un consum de potencia molt baix; per fer-nos una idea, el consum del processador usat, el Cortex-A9 d'ARM, és de 0,25 W, (per cada nucli del processador). Per aconseguir aprofitar tot el rendiment que proporciona l'arquitectura del processador s'integren varis models de programació paral·lels, que engloben el rang de memòria compartida, memòria compartida distribuïda i memòria distribuïda.

Com a objectiu secundari té fer una valoració del potencial del SoC i més concretament del processador Cortex-A9 en la branca de la computació d'altes prestacions. Per assolir-ho, es fa un estudi del mercat sobre les CPUs i les GPUs que hi ha actualment i les que estan projectades fins a 2013.

La memòria està feta per amb l'objectiu de donar una visió global de l'àmbit en que en que es mou, aprofundint més en els temes dels objectius del projecte. Els annexes i el glossari del final de la memòria contenen alguns dels conceptes, tecnologies, etc. que calen saber per entendre millor els temes tractats en la memòria. En la memòria es troven referències a la bibliografia i als enllaços web amb el format [X] per a les referències a la bibliografia i [X*] per a les referècies als enllaços web.

Capítol 2

2 La placa de desenvolupament

La placa de desenvolupament és la “Tegra 200 Series Developer Board”, en aquesta secció de la memòria es donen totes les especificacions necessàries de la placa.

2.1 Especificacions tècniques de la placa

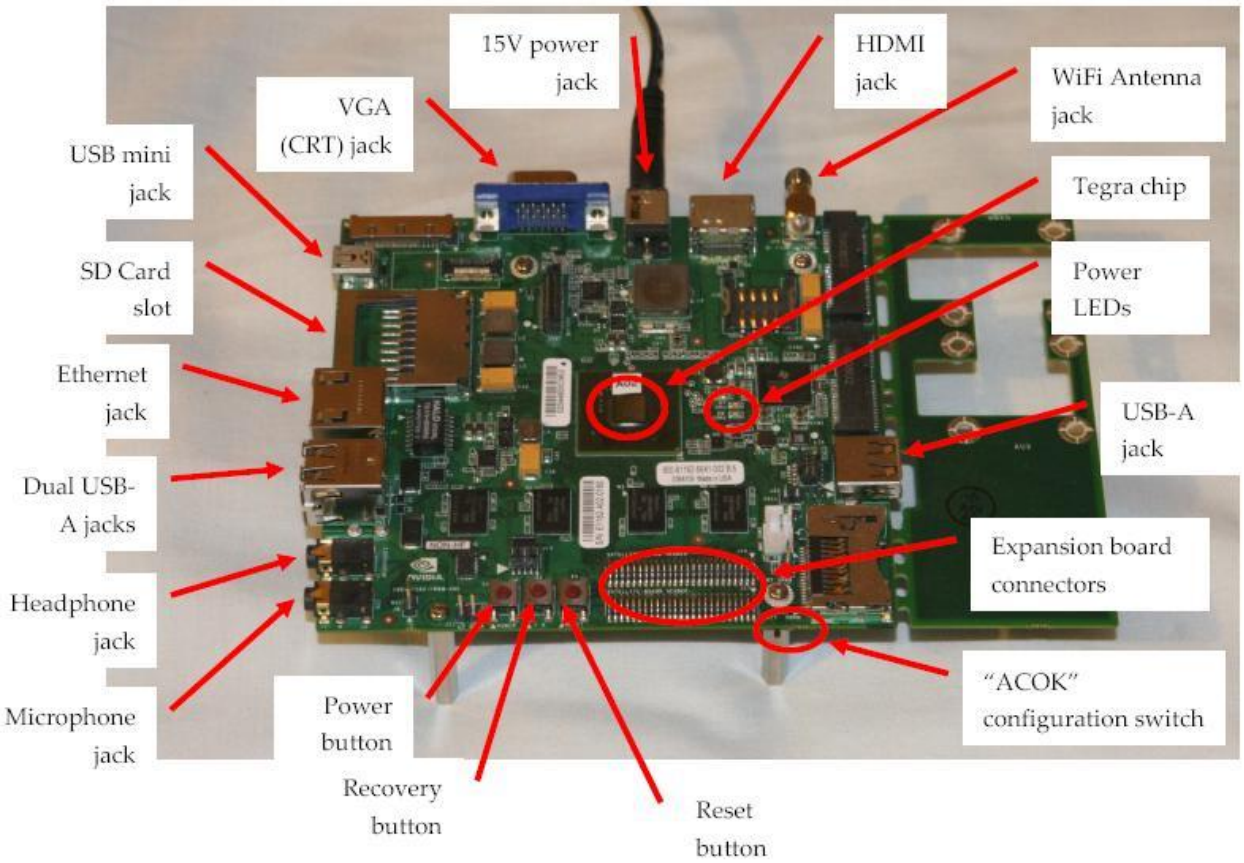

Figura 2.1: Vista de la placa de desenvolupament Nvidia Tegra 200 Series

Com que el xip Nvidia Tegra 250 està pensat per pads, (per exemple, el Toshiba Folio 100), la placa porta tots el ports que tenen els pads i més, [2], [5*].

NVIDIA® Tegra™ 250

Processador Dual-core ARM® Cortex-A9 MPCore™.

Processador gràfic 3D amb OpenGL ES2.0.

Processador de senyals d'imatge amb suport de sensor de fins a 12 Mpixel.

Processador de consum ultra baix.

Sistema d'administració d'energia avançat.

Escalat de freqüència dinàmic.

Múltiples dominis de freqüències de rellotge.

Memòria DRAM

RAM 1GB DDR2–667 (onboard).

Memòria Flash

512MB 8-bit SLC (onboard).

SD/MMC card slot extern: socket 4-bit SD/MMC estàndard permet inserir i treure SD/MMC/SDIO per l'usuari.

SD/MMC card slot intern: combinació de 8-bit MMC i 4-bit SD/MMC socket destinat a ser usat com a dispositiu per arrencar el sistema o per emmagatzematge massiu. No hauria de ser usat com a dispositiu d'emmagatzematge extraïble.

Interfícies de pantalla

Controlador de pantalla dual (LCD integrat + extern).

Single Channel 18 BPP LVDS amb DDC i Backlight Power integrat.

HDMI 1.3 amb resolució suportada fins a 1920 x 1080.

VGA amb suport fins a 1600 x 1200.

Àudio

Wolfson WM8903 L Codec.

Stereo headphone jack.

Connectors de micròfon externs i interns.

Amplificadors de altaveus esquerre/dret.

PCI Express

2 x PCIe Mini-Card slots1 interns.

Slot 0: rutes a SIM/UIM card socket destinat a suportar mòduls de mòdem 3G.

Slot 1: por ser usat per Solid-State drives, una solució WiFi diferent, o altres perifèrics. USB i Ethernet.

USB i Ethernet

3 x USB Type-A Host ports: LAN9514.

USB Host port (PCIe Slot #1): LAN9514.

USB Mini Type-B connector: per mode de recuperació.

Ethernet RJ-45 Jack: LAN 9514.

2 Dual Row Expansion Headers

Interfície de teclat de matriu 16 x 8.

Interfície touchpad amb botos esquerre i dret.

Interfície I2C per sensor de tacte.

Interfície I2C per controlador interface for external incrustat extern.

Botons i interruptors

Botons de POWER, F. R. (Force Recovery) i RESET.

Interruptor de emulació de detecció AC/DC.

Opcions d'alimentació

Adaptador d'AC extern (15V @ 30W).

Bateria de Li de 3 elements 3.7V 2200mAHr (no inclòs): mínim 24WHr.

Controlador de bateria de nivell 2 Level 2.

Wireless

Murata WiFi i mòdul Bluetooth.

Bluetooth: CSR BC6.

802.11g WiFi: Atheros 6002.

Connector d'antena SMA.

Connector CSI Two Data Lane Camera Module

Port per fer debug(JTAG, UART, SPI)

Altres dispositius

Controlador de teclat incorporat: SMSC MEC1308.

Sensor de temperatura: ADT7461ARMZ RL7.

Hub USB amb controlador Ethernet 10/100 SMSC LAN9514.

Transceiver USB Hi-Speed amb interfície 1.8V-3.3V ULPI: SMSC USB3317.

2.1.1 El SoC Tegra 250

El SoC, ("system on a chip"), Tegra 250 és un sistema multiprocessador heterogeni dins d'un mateix xip. La qual cosa vol dir que dins del mateix xip trobem varis processadors amb diferents propòsits.

2.1.2 Vista general del Soc Tegra 250

La següent imatge mostra els processadors dins del xip i la seva ubicació aproximada, [1].

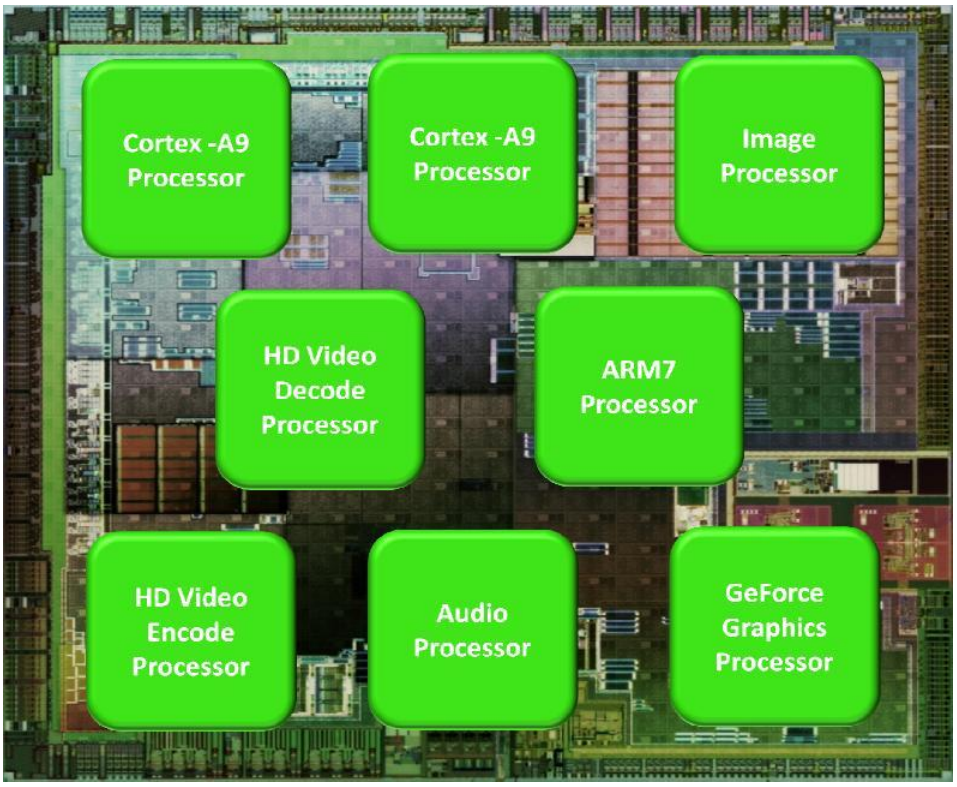

Figura 2.2: Esquema de la ubicació dels processadors dins del SoC Tegra 250

- Dual-core ARM Cortex A9 CPU: És el processador més potent del xip.
- ARM7 Processor: Aquest processador té com a objectiu manegar les funcions del sistema d'administració d'energia.
- Ultra Low Power Graphics Processor (GPU): És la GPU del SoC, és compatible amb OpenGL però, no amb CUDA ni OpenCL.
- HD Video Decode Processor: Corre algorismes orientats a blocks per a video; incloent IDCT, VLD, CSC i bit stream. Capaç d'un "frame rate" de 1080p.
- HD Video Encode Processor: Corre algorismes de codificació de video a 1080p.
- Audio Processor: Manega el processat de senyal de so analògic.
- Image Signal Processor (ISP): Manega aspectes com el balanceig de llum, la millora de les arestes i algorismes de reducció de soroll per donar millores de qualitat fotogràfica en temps real.

2.1.3 Sistema global d'administració d'energia NVIDIA

Uns dels principals atractius del SoC Tegra 250 són poder apagar i encendre els seus components segons el tipus de les aplicacions estiguin corrent en un moment donat i que té esclat dinàmic de freqüència, (DFS), la qual cosa fa que pugui moderar el consum dels components encesos segons la potència de càlcul que les aplicacions necessitin.

2.2 La CPU ARM Cortex-A9

Les seves característiques principals del processador més potent del SoC són:

- Single o multi core a 1Ghz cada nucli, configuracions de 1,2 i 4 nuclis.
- De 2 a 4 instruccions per cicle.
- Arquitectura: ARMv7-A Cortex.
- 0,5 W de consum, (250mW per CPU).
- Àrea ocupada: 4.6 mm2 (incloent DFT/DFM).
- Eficiència energètica: 8 (DMIPS/mW).
- DMIPS: 4000.
- Standard Cell Library: ARM SC12 + High Performance Kit.
- Superescalar.
- Execució fora d'ordre, amb renomenament de registres.
- Pipeline de 8 fases especulatiu.
- Suport per a coherència de cache i SMP.

- Memòries cau de primer nivell, (dades i instruccions separades), de 16, 32 o 64KB associatives amb conjunts de 4.
- Memòria cau de segon nivell unificada per a tots els nuclis, de fins a 8 MB, associativa amb conjunts de 4 conjunts fins a 16 conjunts.

2.2.1 Esquema d'elements

El següent esquema mostra quins components té un processador ARM Cortex-A9; en aquest esquema el processador té 4 nuclis, [2*].

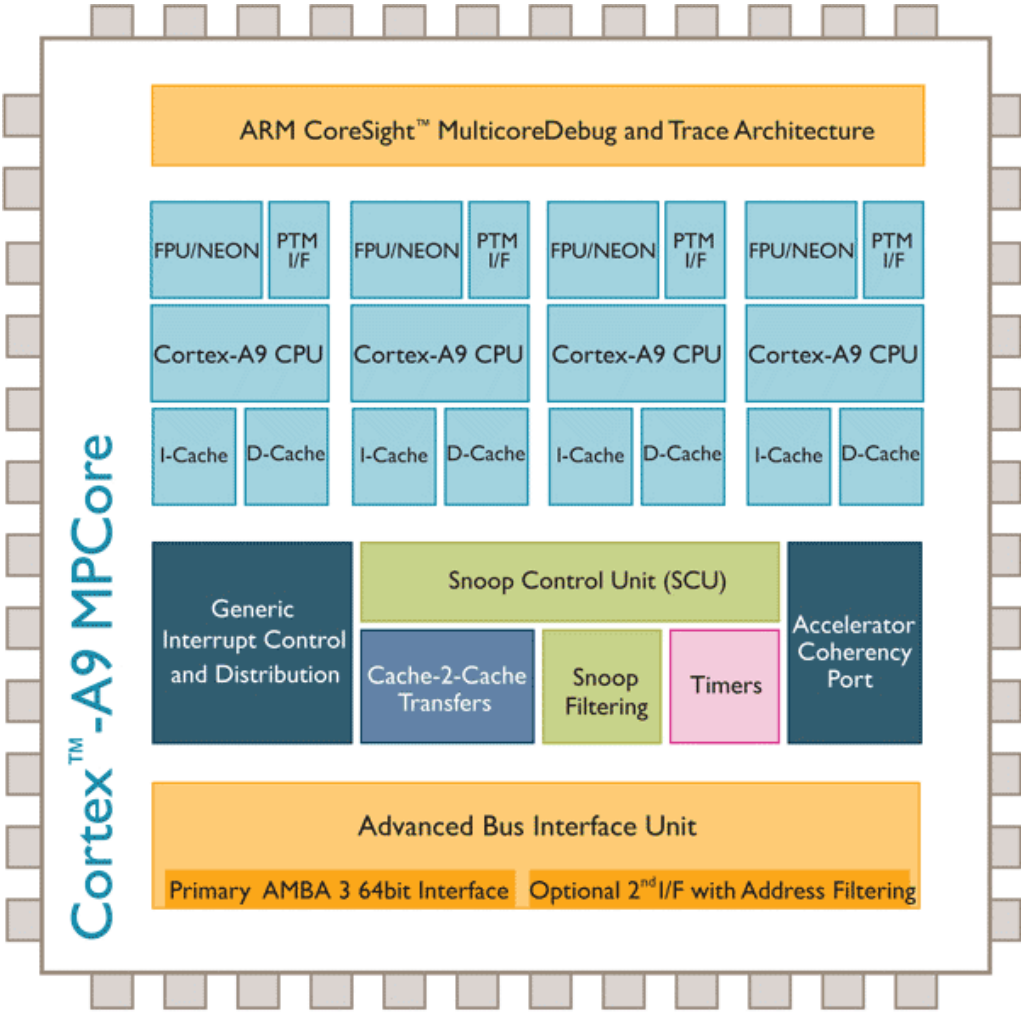

Figura 2.3: Esquema d'un ARM Cortex-A9 MPCore de 4 nuclis

El processador que porta integrat el SoC Tegra 250 només conté 2 nuclis dels fins a 4 que pot tenir el disseny que és pot encarregar a ARM. A més tampoc té les unitats NEON que es veuen en aquest esquema sinó que només té una FPU per a cada nucli.

Anem a veure amb detall que és cada element i perquè serveix:

- Cortex-A9 CPU: És la CPU, on és fan les operacions de càlcul i tot el que fa una CPU pròpiament.
- I-Cache: És la cache de instruccions.
- D-Cache: És la cache de dades.
- FPU/NEON: Són les unitats on és fan les operacions de coma flotant, en la memòria es dedica una apartat per tractar les FPUs amb detall.
- PTM I/F: Program Trace Macrocell, (només per al Cortex-A9). És un tracejador de flux de programa per als processadors Cortex-A9, fa "profiling" amb l'avantatge de que els temps obtinguts durant l'execució amb traça no pateixen variacions.
- Generic Interrupt Control and Distribution: Implementa un controlador d'interrupcions; proporciona un enfocament flexible per les comunicacions inter-processador, l'enrutament i la priorització del sistema d'interrupcions. Suporta fins a 224 interrupcions independents, sota control software, cada interrupció pot ser distribuïda a través de la CPU, prioritzada per hardware, i enrutada entre el sistema operatiu i la capa de software TrustZone, (veure TrustZone al glossari d'aquesta memòria).
- Accelerator Coherency port: Té la finalitat de fer les transferències entre les memòries cau i altres dispositius que no són nuclis.
- Snoop Control Unit (SCU): És el controlador de snooping, gestiona les línies de memòria cau que cada nucli va llegint i escrivint, segons el protocol de coherència que faci servir.
- Snoop Filtering: Serveix per definir regions de memòria que són compartides i necessiten coherència, així es poden diferenciar d'altres regions que no necessiten coherència estalviant temps i consum elèctric.
- Cache-2-cache transfers: Són les transferències que es fan d'una memòria cau a una altre memòria cau.
- Timers: Són temporitzadors que s'usen per prendre decisions en la coherència del sistema.
- Advanced bus interface Unit: És el bus que hi ha entre els processadors i la memòria cau de segon nivell. Poden ser una o dues interfícies de 64 bits. Es capaç de repartir les transferències amb el màxim balanceig de càrrega amb fins a velocitats de transferència de 12 Gb/s. Si hi ha segona interfície, aquesta pot encarregar-se de les transaccions d'un subconjunt d'adreces de memòria. Cada interfície pot oferir diferents ratios de freqüència, millorant l'ample de banda tenint en compte que els processadors tenen escalat de freqüència dinàmic.
- Primary AMBA 3 64-bit Interface: És la interfície d'interconnexió de 64 bits dels components del processador. Veure AMBA al glossari d'aquesta memòria.
- Optional 2nd I/F with address filtering: És la segona interfície opcional que, en cas d'estar, s'encarrega d'unes adreces de memòria en concret.
- ARM Coresight Multicore Debug and trace arquitecture: És un port per depurar i fer seguiment de codi.

2.2.2 Pipeline del Cortex-A9

El pipeline del Cortex-A9 d'un nucli presenta la següent microarquitectura, [4]:

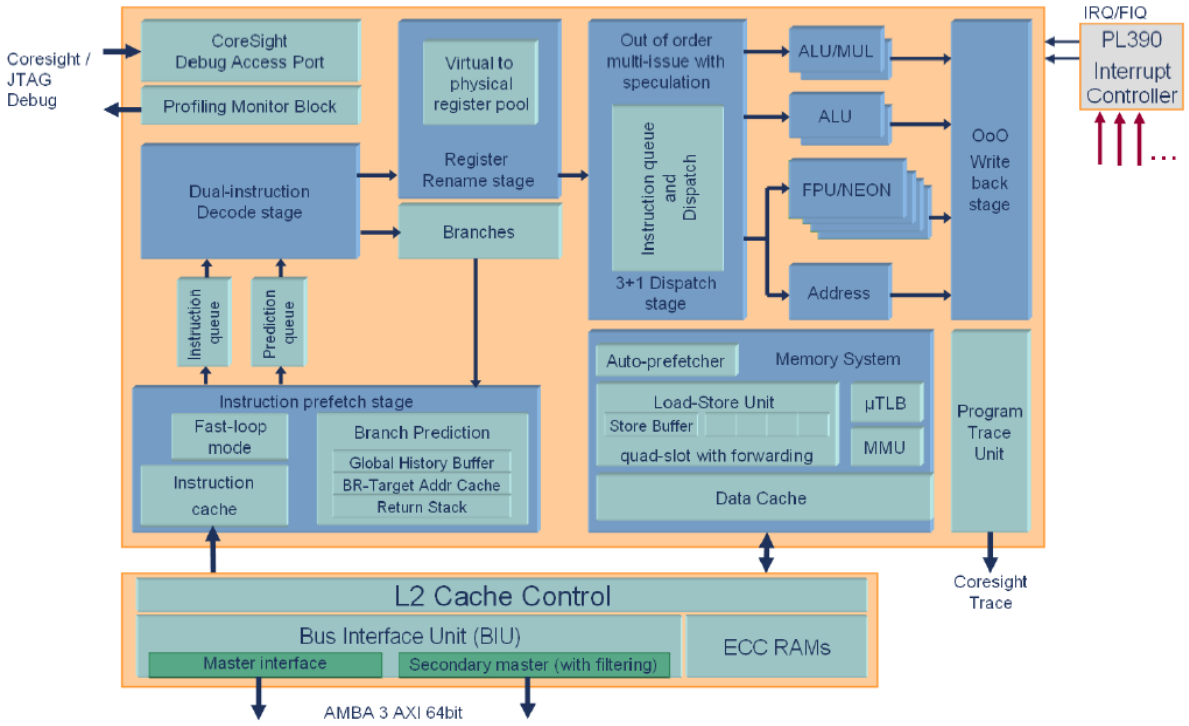

Figura 2.4: Pipeline d'un Cortex-A9 d'un nucli

- Instruction prefetch stage: Té la memòria cau de instruccions de nivell 1, un mòdul per fer bucles curts de manera eficient que estalvia consum i un predictor de salts amb memòria.
- Instruction queue, prediction queue: Són cues d'instruccions a executar.
- Dual instruction decode stage: Ens està indicant que és un processador superescalar. Segons les especificacions cada cicle entren entre 2 i 4 instruccions a la etapa de descodificació no obstant, pot descodificar dos instruccions per cicle.
- Register rename stage: Després de descodificar la instrucció es fa el renomenamant de registres per augmentar el IPC evitant les falses dependències WAW i WAR. Juntament, en aquesta etapa, hi ha un reconeixement dels salts per omplir les estructures de dades del predictor de salts.
- Instruction queue and dispatch: Després del renomenament les instruccions entren a la finestra d'instruccions per poder ser executades fora d'ordre.
- ALU, FPU, Adress, etc: Aquí s'executen les instruccions, com es veu a l'esquema hi ha varies unitats funcionals, la qual cosa indica que pot executar varies intruccions en un mateix cicle.

- OoO write back stage: Com a última etapa està la etapa de write back, amb escriptures fora d'ordre, la qual cosa permet alliberar recursos del pipeline independentment de l'ordre en que el sistema proporciona les dades, [9].
- Memory system: Fora de les etapes del pipeline però, dins del mateix nucli està el sistema de memòria. Conté la memòria cau de dades de primer nivell, el MMU, el TLB, un precarregador de línies de memòria cau de dades i una implementació d'un store buffer.
- L2 cache control, (PL310): Connectat amb la etapa de càrrega, (fetch), i el sistema de memòria, està el SCU que subministra les instruccions i les dades a les memòries cau de primer nivell. Com veiem té un sistema de correcció d'errors ECC en les dades que arriben. Aquest mòdul és compartit per tots els nuclis.
- Coresight debug acces port, profiling monitor block, program trace unit: Són ports i unitats per depurar, monitoritzar i fer seguiment del codi que corre en el nucli.
- PL390 interrupt controller: És el controlador d'interrupcions, veure la secció 2.2.1 d'aquesta memòria per més informació.

2.2.3 Coherència en el Cortex-A9

Pel que fa la coherència, el Cortex-A9 fa ús del mecanisme de coherència snooping, amb un protocol de coherència MESI modificat que suporta transferències “cache-to cache” i intervencions de dades directes. Tot i així, el mecanisme de coherència té més un aspecte de directori, ja que els tags de les línies de memòria cau de cada nucli es troben en una entitat, (SCU), que no està a memòria principal però, està entre els nivells 1 i 2 de memòries cau. EL SCU manté la coherència de la memòries cau de dades i instruccions així com dels MMUs i TLBs.

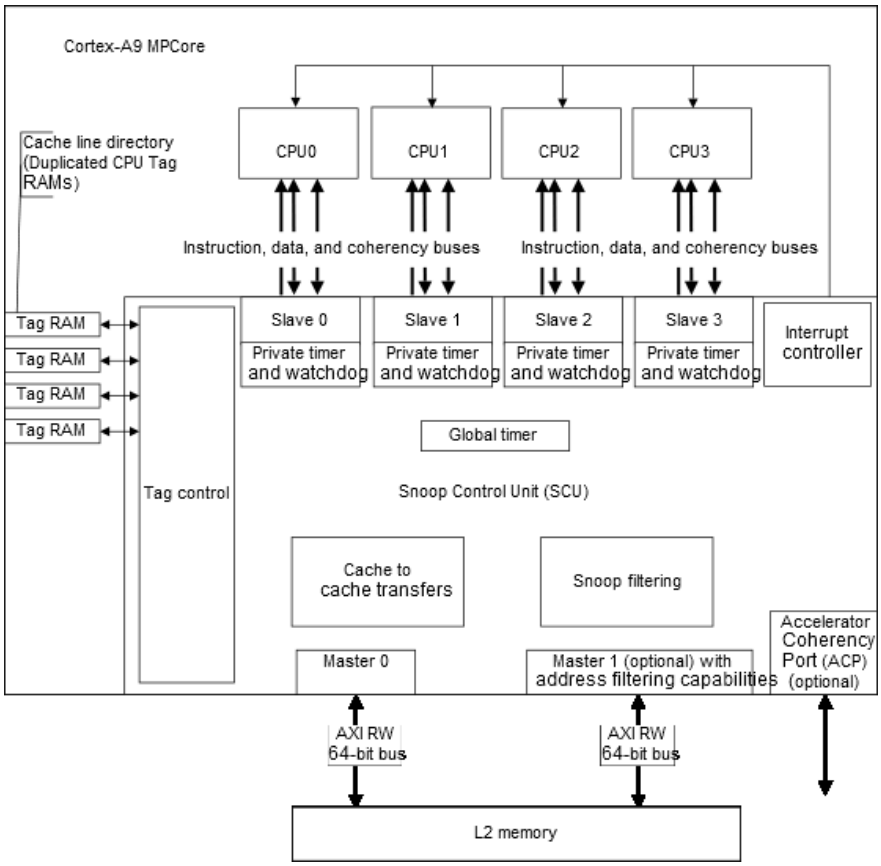

Figura 2.5: Esquema de la coherència en un Cortex-A9 de 4 nuclis

Cada cop que un processador anuncia una lectura o escriptura, ho fa a través d’un canal exclusiu amb el SCU i aquest s’encarrega de fer possible la coherència. D’aquesta manera es pot comprovar si les dades de la memòria cau són correctes sense interrompre la CPU. Una prestació a remarcar que té el SCU és poder filtrar l’accés només a les memòries cau que comparteixen dades amb el snoop filtering guanyant eficiència i estalviant energia.

Una altre de les millores que trobem en aquest processador en quant a mecanismes de coherència està en la integració d’acceleradors en el processador mantenint la coherència de manera més eficient que en amb les metodologies que hi havien prèviament.

En un sistema tradicional amb integració d'acceleradors en el processador, els acceleradors no accedeixen a la jerarquia de memòries cau sinó que només llegeixen i escriuen en memòria principal, [6], [7].

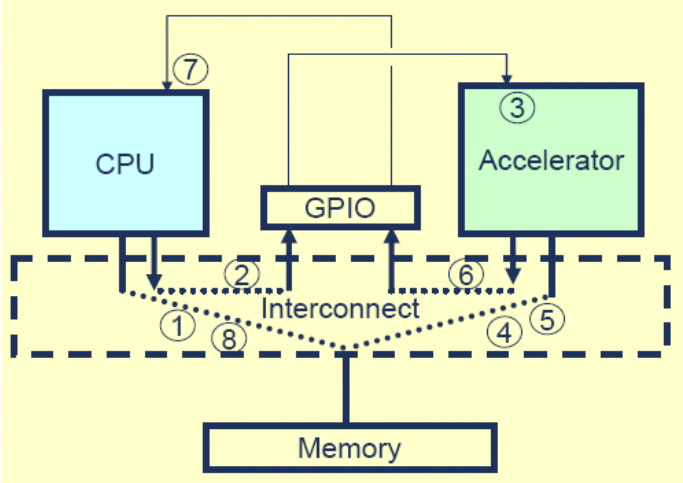

Figura 2.6: Comunicació amb acceleradors tradicional

La seqüència d'esdeveniments és la següent:

1. La CPU buida la memòria cau a memòria principal per fer-la visible a l'accelerador, (flush).
2. Escriu a la GPIO un missatge que l'accelerador pot recollir; indicant que aquest pot anar a accedir a les dades.
3. Es serveix la interrupció a l'accelerador.
4. L'accelerador llegeix de memòria principal.
5. Escriu el resultat a memòria principal.
6. Notifica que les dades estan disponibles.
7. La CPU rep la interrupció.
8. La CPU llegeix el resultat de l'accelerador a la memòria principal.

En el sistema millorat trobem un port accelerador de coherència, (ACP), dins de la unitat de control de snooping, (SCU). S'encarrega de gestionar les transaccions entre els acceleradors i el SCU, [4], [6], [7] .

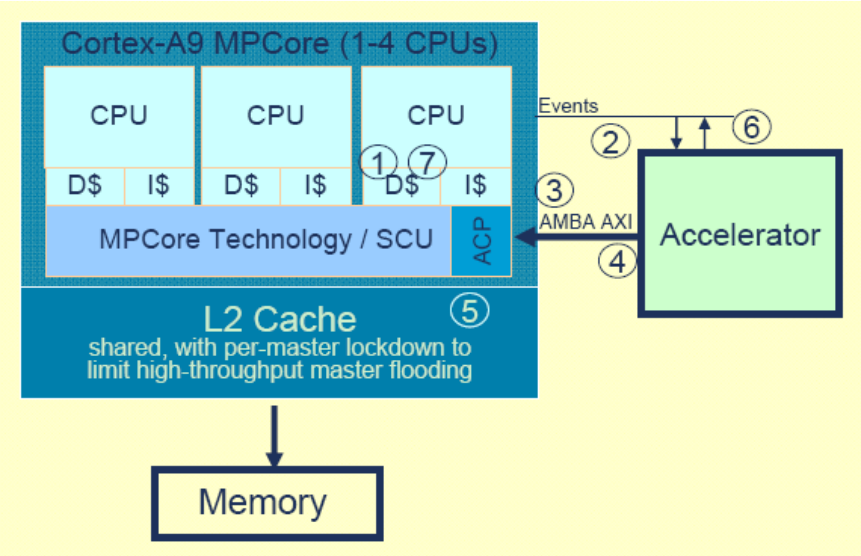

Figura 2.7: Comunicació amb acceleradors millorada

La seqüència de passes per a una comunicació entre un accelerador i el nucli multiprocessador és:

1. La CPU deixa les dades a la memòria cau.
2. El proper cicle notifica a l'accelerador comprovar i processar les dades.
3. L'accelerador demana llegir les dades, aquestes han de retornar d'algun lloc de la jerarquia de memòria. De nivell 1, 2 o memòria principal.
4. L'accelerador demana escriure el resultat, la coherència a nivell 1 és assegurada.
5. Probablement estigui configurat per escriure a nivell 2.
6. L'accelerador avisa al nucli que ha de llegir les dades que ja les té disponibles.
7. La CPU demana la lectura, probablement estigui a nivell 2.

A grans trets, es tracta de que els acceleradors tinguin accés a la jerarquia de memòries cau, i així reduir el cost extra, (overhead), d'haver de fer buidats, (flush), de cache i les latències extra que comporta sincronitzar la CPU amb l'accelerador a través d'un element d'entrada sortida general.

2.2.4 Superfície ocupada per element

En l'esquema de sota veiem la superfície ocupada en transistors de tot el "die". La versió amb optimització de consum elèctric està fabricat amb el procés TSCM de 40 nm , ocupa 4,6 mm2 i està fet amb silici lent 125 C Tj.

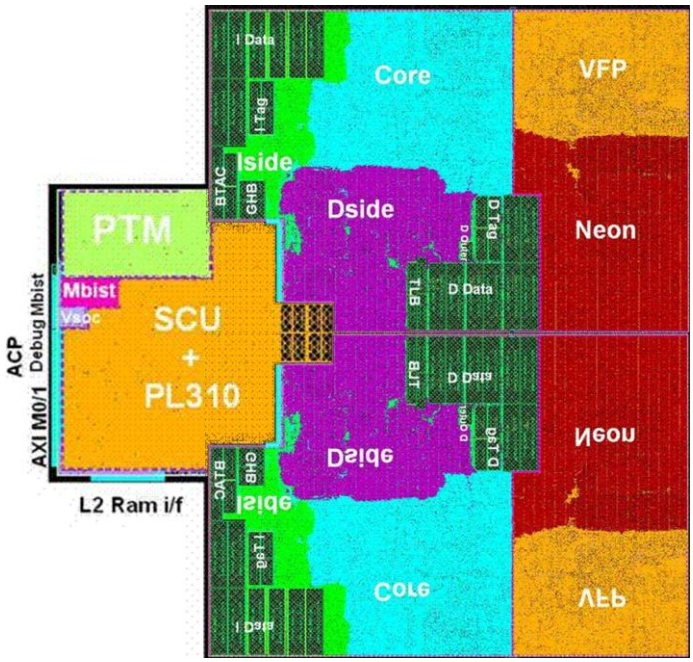

Figura 2.8: Vista "floorplan" d'un Cortex-A9 de 2 nuclis

Veiem els mòduls de VFP i NEON que com hem comentat són opcionals en cada nucli. Les memòries cau de primer nivell poden tenir una mida diferent per cada nucli. Els components que més ocupen són els nuclis d'execució, el SCU i Dside.

El Cortex-A9 té la capacitat d'apagar els seus mòduls segons si s'estan utilitzant o no, per exemple, pot apagar les VFP, les unitats NEON o fins i tot memòries cau de nuclis que s'estiguin utilitzant parcialment.

2.2.5 Unitats vectorials

El processador ARM Cortex-A9 del SoC Tegra 250 disposa d'una unitat FPU anomenada VFP, (vector floating point), per a cada nucli, per contra no disposa de unitats Neon. Recordem que els dos tipus d'unitats vectorials són opcionals en un Cortex-A9.

2.2.5.1 VFP floating point unit

La unitat de càlcul de coma flotant VFP, pot fer operacions en meitat, simple i doble precisió; segons la norma IEEE 754. Té una llibreria software de suport.

La versió de la VFP que conté el Cortex-A9 és la VFPv3. Està implementada amb dos registres de 16 bits, el que es correspon amb el nom de VFPv3-D16.

Algunes operacions que esdevenen en pocs casos o que són molt complexes no són suportades pel hardware VFP, aquests casos són tractats per software. Les operacions per software són notablement més lentes que les executades per hardware.

Les unitats vectorials estan situades en la etapa d'execució del pipeline no obstant, les instruccions que s'hi han d'executar queden encuades en una cua dedicada a la VFP del nucli.

El document Cortex-A9 "Floating-Point Unit Revision" de la bibliografia és un manual tècnic de referència sobre com usar les unitats vectorials que trobem al Cortex-A9, [8]. Destaco les següents característiques:

Antic mnemotècnic assembler d'ARM	Mnemotècnic UAL, (unified assembler lenguage)	Simple precisió			Doble precisió		
		Throughput	Latència		Throughput	Latència	
			Fwd	Wck		Fwd	Wck
FADD	VADD	1	4		1	4	
FSUB	VSUB						
FMUL	VMUL	1	5		2	6	
FMAC	VMLA	1	8		2	9	

Taula 2.1: Throughputs i latències de les VFP

- VADD: suma dos nombres de simple o doble precisió.
- VSUB: resta dos nombres de simple o doble precisió.
- VMUL: multiplica dos nombres de simple o doble precisió.
- VMLA: multiplicació acumulativa; multiplica dos nombres i suma el resultat a un altre nombre; (s'utilitza molt en la multiplicació de matrius).

On el throughuput és el nombre de cicles que una altre instrucció igual a la anterior a d'esperar per entrar en la unitat d'execució, en aquest cas la VFP. Latència és el nombre de cicles que les dades triguen a ser disponibles per una altre operació; Fwd és rellevant per als riscos de dades "read after write", (RAW), Wbk és rellevant per als riscos de dades "write after write", (WAW).

Ara estem en condicions de fer una mica d'aritmètica i aproximar els FLOPS de les unitats vectorials en una multiplicació d'una matriu grossa.

Considerant només que s'han d'executar les instruccions de multiplicació acumulativa.

1 operacions de coma flotant de doble precisió per cicle * 800 MHZ * 2 nuclis = 1600 MFLOPS.

Com que el consum del processador és de 0,5 W, el Cortex-A9 té un rendiment de 3,2 GFLOPS/W.

Tenint en compte que la manera més òptima d'aconseguir FLOPS fins ara era usant GPUs, per exemple amb una NVIDIA Tesla C2050, amb un rendiment de 2 GFLOPS/W, ens resulta que el Cortex-A9 a més de tenir els avantatges per programar dels processadors de propòsit general és en realitat la plataforma amb més rendiment GFLOPS/W.

Està clar que per fer ús dels dos nuclis, (cada un amb una unitat VFP), haurem de paral·lelitzar el nostre codi, per exemple, fent ús de threads.

Capítol 3

3 Instal·lació de Linux i verificació del funcionament

Existeix una guia en angles per instal·lar Linux a la placa a la web, http://developer.nvidia.com/tegra/downloads, no obstant, a continuació faig un tutorial resumit i amb les comandes que s'han de fer servir, [3].

Per instal·lar Linux a la placa el primer que hem de fer és anar a la web i descarregar el pack Linux For Tegra (L4T) Release, en aquest tutorial vaig usar la versió amb el kernel 2.6.32, [1*].

3.1 Preparació de la placa

Hem de connectar a la placa el següent:

- Un ratolí USB.
- Un teclat USB.
- Un monitor de connector VGA o HDMI.
- La font d'alimentació.
- Connexió cablejada o wireless per instal·lar paquets o per connectar-nos per ssh (opcional).

3.2 Requeriments al PC host

El PC d'es d'on construirem el sistema operatiu ha de complir els següents requisits:

- Tenir Ubuntu 9.X o 10.x instal·lat.
- Port sèrie de 9 pins (opcional, no l'usarem en aquest tutorial).
- Un cable null-modem sèrie de 9 pins (opcional, no l'usarem en aquest tutorial).
- Si el nostre PC té instal·lat un Ubuntu de 64 bits, haurem d'instal·lar el suport de runtime per a 32 bits.
    - sudo apt-get update
    - sudo apt-get install ia32-libs

3.3 Preparació del SO en el dispositiu d'emmagatzematge

1. Moure el fitxer linux_for_tegra_os_pack_<RELEASE_NUMBER>.run al directori en el que vulguem treballar.
    - mv linux_for_tegra_os_pack_<RELEASE_NUMBER>.run "path directori que volem).
2. Instal·lar el paquet.
    - bash linux_for_tegra_os_pack_<RELEASE_NUMBER>.run
3. Llegir i acceptar les condicions de llicència
4. Moure'ns al subdirectori creat .
    - cd linux4tegra
5. Especificar el path absolut al path linux4tegra.
    - export L4TROOT=${PWD}
6. Connectar el dispositiu d'emmagatzemament al PC host. Aquest pot ser un USB, una targeta SD o fins i tot un disc dur USB. En aquest tutorial s'ha fet servir un USB no obstant, recordem que una targeta SD o un disc dur USB haurien de permetre velocitats més altes d'accés a disc.
7. Si el nostre SO ens munta el dispositiu per defecte l'haurem de desmuntar. El path on desmuntar varia depenent del sistema no obstant, poso un exemple.
    - sudo umount /dev/sdc
8. Particionar el dispositiu. Ho podem fer des de intèrpret de comandes amb la comanda fdisk, (fer man fdisk per saber com funciona), o des de l'entorn gràfic. Per fer-ho des de l'entorn gràfic s'ha de fer:
    - Anar a sistema, administració, utilitat de discs.
    - Seleccionar el dispositiu que volem formatejar.
    - Seleccionar el tipus de partició, ext3.
    - Escriure la etiqueta que volem que tingui el dispositiu.
    - Marcar que és arrancable.
    - Prémer el boto aplicar per formatejar.
9. Un cop amb el dispositiu formatejat el muntem, si tenim automuntatge només cal treure el dispositiu i tornar-lo a endollar, sinó podem fer servir la comanda mount.
10. El següent pas és desempaquetar el pack que hem creat abans en el punt 2.
    - ./unpack_targetfs.sh
11. Ara copiem els fitxers del SO en el lloc que correspon al sistema de fitxers.
    - ./apply_14t.sh
12. En aquest pas copiem el sistema de fitxers al dispositiu USB.
    - Sudo cp –rp _out/targetfs/* /mnt/disk

13. Finalment, desmuntem el USB i l'endollem a la placa.

3.4 Flash de la placa

1. Posar la placa en estat de recuperació.
2. Connectar la placa al PC host amb el cable USB-to-mini-USB.
3. Córrer el script flash_14.sh, segons si fem servir el port USB o el port SD (usb, sdmmc) i segons si fem servir VGA o HDMI (crt, hdmi). Exemples:
    a. ./flash_14.sh usb crt
    b. ./flash_14.sh sdmmc hdmi
4. Desconnectar el cable USB.

3.5 Primera arrencada

- Quan ens demani l'usuari i el password entrar: ubuntu, ubuntu.
- Tenir en compte que la disposició de les tecles és segons el teclat americà.
- Si volem connexió podem fer servir:
    - sudo dhclient
    - O sinó volem fer servir cada cop que bootem el sistema la comanda anterior; obrir el fitxer /etc/network/interfaces amb el vi i editar el següent:
        - auto usb0
        - iface usb0 inet dhcp
- Si volem connexió sense fils, recomano fer un script amb les següents línies:
    - sudo insmod /system/lib/hw/wlan/ar6000.ko
    - sudo ifconfig wlan0 up
    - sudo iwconfig wlan0 essid ESSIDX (on ESSIDX és el nom del nostre essid).
    - sudo dhclient

3.6 Utilitats per la placa

- Per monitoritzar l'estat del xip Tegra 2 podem fer servir l'aplicació tegrastats. Recomano veure el manual especificat en la secció 3 per saber que és cada paràmetre que ens retorna l'aplicació.
    - ./tegrastats <retard en segons>

- Per monitoritzar més concretament l'estat dels nuclis del processador Cortex-A9 podem descarregar la següent aplicació:
    - sudo apt-get install sysstat
    - man mpstat
- Per carregar X11 actualment hi ha poc suport. Hem de fer:
    - Instal·lar algun gestor de finestres no gaire complex. Per exemple:
        - sudo apt-get update
        - sudo apt-get install metacity
        - Podem optar per un altre gestor: sudo apte-get install fluxbox
    - Xinit -- -layout CRT &
    - Escriure metacity o fluxbox
    - Notació: Les resolucions no estan ben implementades així com la posició del cursor real del ratolí respecte la que es veu en pantalla.
- Per treballar des de ssh només cal fer el següent des del PC que vulguem treballar:
    - ssh ubuntu@192.168.1.X, (on X és la direcció que ens hagi assignat el DHCP).
    - password: ubuntu

3.7 Verificació del funcionament

El que s'ha de fer abans de tirar endavant amb un component és veure que realment suporta el tipus de computació que esperem d'ell. És per això que en aquest punt he verficat el funcionament del processador ARM Cortex-A9 de dins del SoC Tegra 2, que és el component que més ens interessa de tot el SoC.

3.7.1 Rendiment dels components del sistema per defecte

Per saber quin és el rendiment dels components de la placa he fet servir el programa "tegrastats" que ve per defecte en el Linux per Tegra (L4T).

A continuació mostro els paràmetres dels components més representatius de la placa; aquests paràmetres es corresponen quan tot el sistema està acabat d'arrencar sense cap programa d'usuari corrent:

Memòria RAM total usada en el moment.	28 MBytes.
Memòria RAM disponible per a aplicacions.	629 MBytes.
Memòria de la GPU usada.	8 MBytes.
Memòria de la GPU disponible en total.	64 MBytes.
Percentatge d'ús del nucli 1 que està sent usat relatiu als megahertzs que està corrent la CPU en el moment.	10%.
Percentatge d'ús del nucli 2 que està sent usat relatiu als megahertzs que està corrent la CPU en el moment.	Nucli apagat.
Freqüència de la CPU a la que està treballant en el moment.	50 Megahertzs.

Taula 3.1: Rendiment dels components del sistema per defecte

(Notació: la memòria RAM disponible varia entre versions de SO).

3.7.2 Rendiment màxim dels components del sistema

Per saber quina és la màxima freqüència a la que poden treballar els components de la placa fem servir la comanda "./tegrastats –max".

Percentatge d'ús del nucli 1 que està sent usat relatiu als megahertzs que està corrent la CPU en el moment.	0%.
Percentatge d'ús del nucli 2 que està sent usat relatiu als megahertzs que està corrent la CPU en el moment.	Nucli apagat.
Freqüència de la CPU a la que està treballant en el moment, en megahertzs.	1.000 Megahertzs.

Taula 3.2: Rendiment màxim dels components del sistema

Observació: els altres components que no s'han analitzat també escalen la seva freqüència al màxim.

3.7.3 Proves amb múltiples processos

Per comprovar que el processador té suport per a executar varis processos, concretament dos a la vegada, ja que hi ha dos cores; el que es fa es fer un petit programa que faci uns quants processos amb un simple fork, i observar com els dos nuclis treballen en la creació d'aquests processos.

3.7.3.1 Codi Programa

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 1.

3.7.3.2 Monitorització

Efectivament, en la figura 3.1 veiem monitoritzant amb el script ./tegrastats, com al executar el programa els dos nuclis treballen a la vegada. Si imprimim per la sortida estàndard les creacions i execucions de processos obsevarem que per cada procès creat aquest s'executa immediatament, això és degut a que el càlcul que es fa en la rutina del procès és tant petit que acaba i es crea un altre procés sense necessitat de que possiblement acabi el seu quantum de temps al processador.

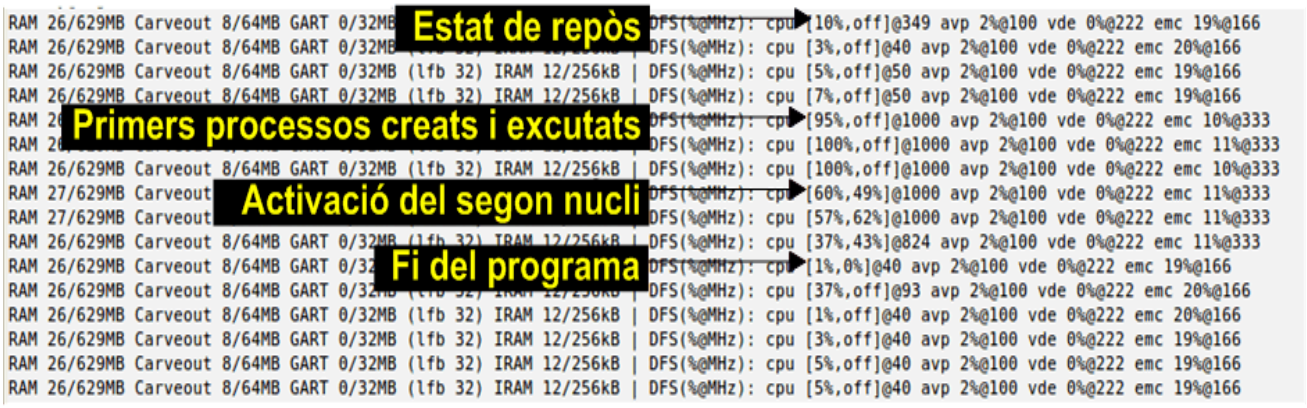

Figura 3.1: Monitorització de la creació de processos

3.7.4 Proves amb múltiples tasques

Com en el cas de múltiples processos, per a comprovar que el processador té suport per a executar varies tasques a la vegada, concretament dues, ja que hi ha dos nuclis sense "simultaneous multithreading", (SMT), el que es fa és fer un petit programa que creï uns quants threads amb la llibreria de pthreads. Hem d'observar que els dos nuclis tenen una càrrega de treball.

3.7.4.1 Codi Programa

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 2.

3.7.4.2 Monitorització

En la captura veiem com inicialment un nucli està encès i l'altre apagat; al començar a executar el programa un nucli fa tota la feina de crear el threads, després el dos nuclis executen threads, amb el mètode do_nothing(). Al acabar el programa queda només un nucli executant el SO com abans de començar a executar el programa.

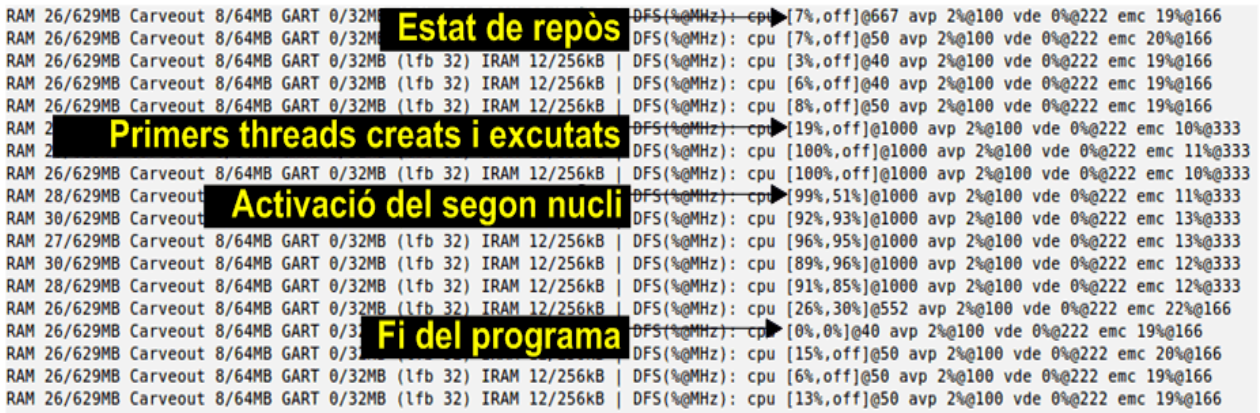

Figura 3.2: Monitorización de la creación de threads

3.7.4.3 Coherència i consistència amb múltiples fils d'execució

Un cop veiem que els dos nuclis treballen al executar el programa que crea les tasques, amb múltiples tasques, després també hem de tenir en compte que el sistema de nuclis ha de tenir coherència i consistència. El fabricant ja ens diu que el processador està preparat però, si en volem estar segurs una manera fàcil és executar una quants programes en seqüencial, després en paral·lel i veure que els resultats són els mateixos. Per asegurar més també podem comprovar que els resultats són els mateixos que en algun lloc on sapiguem que funcionen la coherència i la consistència.

Capítol 4

4 Mesures de rendiment

Abans de posar-se a treballar amb un processador o en aquest cas amb un sistema en un xip per fer valoracions, hem de fer mesures de rendiment bàsiques i conegudes per la comunitat de gent que es dedica a la computació, per tal de poder comparar.

Dos dels paràmetres més determinants per mesurar el potencial de càlcul d'un sistema són els milions d'operacions en coma flotant que pot fer en un segon i l'altre és l'ample de banda que té un sistema des del processador fins a la memòria principal.

4.1 Procés de compilació

Per fer les mesures de rendiment, convé que el nostre processador rendeixi al màxim possible. Per aconseguir-ho hem de compilar els benchmarks i programes que vulguem córrer amb la màxima eficiència que ens permeti el compilador. En el cas dels benchmarks, haurem d'informar-nos de com respectar les seves regles per executar-lo.

El compilador que he usat ha estat el Gcc en la seva versió 4.3.3. Vegem les opcions de compilació, ("flags"), que ens permet usar per al nostre processador Cortex-A9: És una llista extensa que es troba a http://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html. A continuació exposo les opcions més interessants i si funcionen o no.

- -march=armv7-a; funciona.
- -march=armv7; no funciona.
- -mfpu=vfp; funciona.
- -mfpu=neon; funciona però, d'aquesta opció no en podrem fer ús, ja que la nostre versió del Cortex-a9 no té unitats MPE NEON.
- -mfp=vfpv3-d16; funciona només si s'especifica també –mfpu=vfp.
- -O3; funciona.
- -mfloat-abi=softfp; funciona, és la opció per defecte.
- -mfloat-abi=softfp; funciona.
- -mfloat-abi=hard; no funciona.
- -mcpu=cortex-a9, no funciona, està previst que estigui suportat per el Gcc 4.4.1.

4.1.1 Flags de compilació òptims

Per cercar els flags de compilació òptims cal provar les combinacions de flags que creiem que tindran un rendiment més alt. Per fer-ho, he usat un programa senzill de producte escalar amb un thread i 1MB de dades a cadascun dels dos vectors, [9*], [10*], [1*], [12*].

Després de provar una sèrie de combinacions executant el producte escalar he pogut comparar el rendiment de la combinació de flags més eficient amb la menys eficient.

- gcc -pthread –O3 –march=armv7-a –mfp=vfpv3-d16 –mfpu=vfp –mfloat-abi=softfp –o dotprodserial_all dotprod_serial_all.c
    - Mitjana de 4 execucions a 5,73 segons de temps real.
- gcc -pthread –o dotprodserial_all dotprod_serial_all.c
    - Mitjana de 4 execucions a 17,90 segons de temps real.

Speed-up de 3,12; tanmateix hem de tenir en compte que durant execucions més llargues, (al cap d'uns 25 segons), la velocitat del rellotge baixa en una relació de 1,62 vegades per la resta d'execucions. He observat que el Cortex-A9 puja de freqüència fins a 1 GHZ els primers segons d'executar aquesta aplicació per després baixar a 618 MHZ, on la mitjana de les dues freqüències és 809 MHZ, que és la freqüència que està especificada pel fabricant. Veurem aquest comportament amb més detall més endavant.

4.2 Especificacions rellevants dels processadors usats

	Cortex A-9	Core 2 Duo	Itanium 2 Montecito	Pentium 4
Potencia consumida	0,5 W	35 W	104 W	82 W
Freqüència	800 MHz	2,1 GHz	1,6 GHz	3,2 GHz
Arquitectura	Dos nuclis , OoO, Superscalar, register renaming, speculative execution.	Dos nuclis coarse grain threaded, Superscalar, 32 bits.	Dos nuclis coarse grain threaded, VLIW, 32 bits.	Arquitectura: Un nucli amb hyperthreading, OoO, SMT, 32 bits.
Versió Gcc	4.3.3	4.4.3	X	4.4.3
Versió Linux	Ubuntu 9.04	Ubuntu 9.10	X	Ubuntu 10.04

Taula 4.1: Especificacions rellevants dels processadors usats

4.3 Ample de banda de memòria

L'ample de banda de memòria és el coll d'ampolla dels processadors d'avui en dia, ja que el potencial de càlcul del processadors ha pujat durant molts anys de manera molt més pronunciada que la capacitat de les memòries per transferir dades. És per això que els processadors convencionals tenen jerarquia de memòria. Altres casos com les targes gràfiques o processadors vectorials, usen altres tècniques per solucionar el problema que no tractarem en aquesta memòria.

4.3.1 Stream benchmark

El "Stream benchmark" és un benchmark prou conegut, a la seva URL podem trobar des de la seva descripció fins comparacions de sistemes que han passat el benchmark, http://www.cs.virginia.edu/stream/. És un benchmark sintètic, que serveix per mesurar l'ample de banda de memòria principal, [3*].

En la taula de sota veiem quines són les mètriques que fa servir el stream benchmark.

Nom	Kernel, (operació)	Bytes/iteració	FLOPS/iteració
Copy	a(i) = b(i)	16	0
Scale	a(i) = q*b(i)	16	1
Sum	a(i) = b(i) + c(i)	24	1
Triad	a(i) = b(i) + q*c(i)	24	2

Taula 4.2: Operacions i mètriques del stream benchmark

Per fer-lo servir, ens cal definir en el fitxer stream.c, (en cas d'usar la versió en C), la mida de les cadenes de manera que sempre siguin més grans que la memòria cau d'últim nivell, concretament 4 vegades més com a mínim, així es força la interacció amb la memòria principal. Un cop sobrepassada aquesta mida no és gaire important si la sobrepassem massa.

Per exemple, si la nostra memòria cau L2 és de 256 KB, haurem de declarar que volem el vector de 128K elements; ja que cada element conté 8 bytes.

La compilació depèn del processador amb que estiguem treballant. Si és un processador d'un sol nucli només caldrà compilar de manera normal tanmateix, si volem mesurar un processador amb varis nuclis, com que en una execució usant els dos nuclis aquests dos voldran accedir a memòria hem de compilar el programa amb el flag d'OpenMP.

Si per exemple volem mesurar un node SMP, aleshores haurem de fer la mida de les cadenes tant grossa com l'últim nivell de memòria cau multiplicat pel nombre de processadors del node multiplicat per 4. A més clar està, de compilar amb el flag d'OpenMP.

4.3.1.1 Resultats i valoracions

Els resultats estan expressats en MB/s. Per al Cortex-A9 vaig compilar per fer les proves amb els dos nuclis mentre que amb el Core 2 Duo, al estar treballant amb un Linux virtualitzat les proves les vaig fer amb un nucli. Per al Itanium 2 també vaig compilar per a dos nuclis i amb el Pentium 4 amb 1 nucli ja que només en té un.

	Cortex-A9	Core 2 Duo	Itanium 2	Pentium 4
Copy:	610	2571	4834	1670
Scale:	309	2537	4841	1671
Add:	440	2836	5172	1995
Triad:	308	2859	5200	1994

Taula 4.3: Resultats de l'execució del stream benchmark

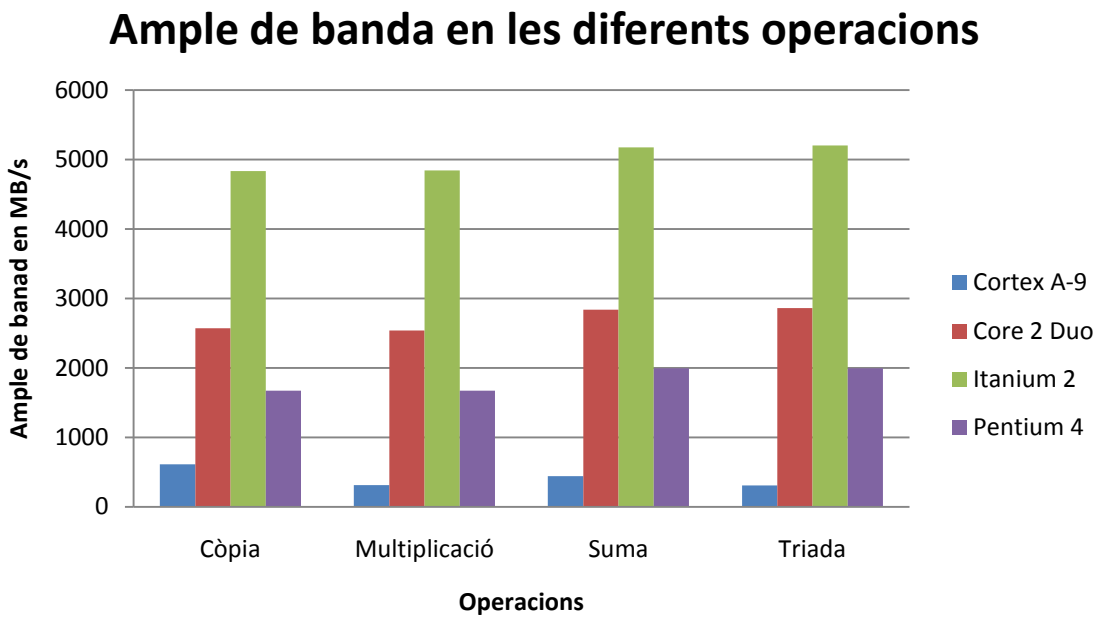

Figura 4.1: MB/s que s'assoleixen en cada operació

Ara calcularem els MB/s que necessitaria el processador i els MB/s que pot sumministrar la memòria RAM que porta la placa.

Agafem la operació de suma del Cortex-A9, sabem que té un throughput de 1 i que fa una operació de suma per tant, calculem els MFLOPS potencials amb la suma:

- 1 operació de coma flotant de doble precisió per cicle * 800 MHZ * 2 nuclis = 1600 MFLOPS.

El que vol dir que necessitaria que la memòria RAM suministres dades amb una velocitat de transferència de:

- 1600 MFLOPS (MFLOP/s) x 24 bytes/FLOP = 38400 MB/s.

Tenint en compte que el rendiment aproximat que aconsegueixen entre els 4 mòduls de RAM de 256 MB cadascun és de 441 MB/s per a aquesta operació. Resulta que encara tenint jerarquia de memòria, la memòria principal està fent rendir 87 vegades menys el rendiment que podríem treure al processador. És a dir, que el processador ens està entregant menys FLOPS del que pot entregar, conretament ens dona:

- 441 MB/s x 1/24 FLOP/Bytes = 18,38 MFLOPS, (MFLOP/s).

No obstant, no ens hem de prendre aquest rendiment estrictament com el rendiment a la pràctica del processador dins de la placa. De fet, executant altres codis per calcular el rendiment en MFLOPS, fent el mínim ús de les transferències de memòria, he arribat a aconseguir 250 MFLOPS amb un nucli usant només operacions de suma i multiplicació amb coma flotant de doble precisió.

4.4 Càlcul general

4.4.1 Dhrystone Benchmark

El Dhrystone benchmark, [13*], és un benchmark sintètic que prova el rendiment en general d'un nucli d'un processador. Conté moltes instruccions simples, crides a procediment i condicions, i poques instruccions de coma flotant i bucles. No realitza crides a sistema. Usa poques variables globals i executa operacions amb punters. Està composat de 12 procediments inclosos en un bucle, el qual no es pot variar la seva mida. Està composat per un 53% d'instruccions d'assignació, un 32% d'instruccions de control i un 15% de crides a procediment. Dhrystone és compacte, (no té més de 1,5 KB); per ser tan petit, el Dhrystone cap completament en la memòria cau interna, d'aquesta manera no mesura la resta del sistema. El Dhrystone compara el rendiment del processador usant una màquina de referència: la VAX 11/780, és la màquina que corre a 1 MIP (fa 1757 Dhrystones per segon).

Iteracions Dhrystone per segon = rellotge del processador * nombre de passades / temps d'execució. Per a que el resultat sigui vàlid del codi Dhrystone ha de ser executat durant almenys dos segons.

El benchmark presenta alguns problemes ja que és molt susceptible a la capacitat del compilador per compilar òptimament per a l'arquitectura donada. És per això que el més adient és compilar en totes les màquines amb els mateixos flags.

La versió usada ha estat la 2 que elimina la possibilitat de que el compilador elimini codi mort; alguns fabricants encara usen la versió 1 donant resultats més elevats. Entre les regles per a passar el benchmark en una màquina està compilar amb unitats de compilació separades per evitar que el compilador optimitzi unint informació, (variables, registres, etc.), de les diferents unitats de compilació. És permet fer inlining de funcions de biblioteca però, no inline de crides a funcions del propi benchmarck. Per la qual cosa en cas d'usar el flag -03 s'hauria d'afegir el flag –fno-inline.

Com tots els benchmarks el Dhrystone també té punts crítics. Un temps molt considerable es gasta en dues operacions; una d'assignació de caràcters i l'altre de comparació de caràcters, cosa que fa al test molt dependent de la manera en que aquestes operacions són executades. El Dhrystone en aquest sentit mesura més aviat el rendiment del compilador i les seves biblioteques que el processador en si mateix. És molt difícil que aplicacions útils tinguin una proporció tant important d'ús de biblioteques a C.

Tanmateix, altres benchmarks com el Whetstone o el Linpack també tenen punts crítics. El primer depèn massa de la velocitat del les funcions matemàtiques, (sinus, sqrt, etc.), mentre que el segon és sensible a l'alineament de dades per algunes configuracions de memòria cau. Passa un 90 % del seu temps en la rutina DAXPY de la biblioteca BLAS, la qual fa una operació Y() := Y() + a * x(), (també coneguda com MULADD), que és la operació fonamental per a la multiplicació de matrius.

4.4.1.1 Compilació i execució

gcc –O3 –DNO_PROTOTYPES=1 –DTIME –c –fno-inline -o dhry_1.o dhry_1.c

gcc –O3 –DNO_PROTOTYPES=1 –DTIME –c –fno-inline -o dhry_2.o dhry_2.c

gcc –O3 –DNO_PROTOTYPES=1 –DTIME –fno-inline dhry_1.o dhry_2.o –o dhrystone

./dhrystone (222.222.222 voltes)

4.4.1.2 Resultats i valoracions

	Cortex A-9	Core 2 Duo	Itanium 2	Pentium 4
Dhrystones per segons	1782943,5	6025867,2	3745553,4	4837017,5
Microsegons per una execució de Dhrystone	0,55	0,2	0,3	0,2
DMIPS	1044,76	3429,63	2131,79	2752,99
DMIPS/W	2089,52	97,98	20,49	33,57

Taula 4.4: Resultats d'executar el Dhrystone benchmark

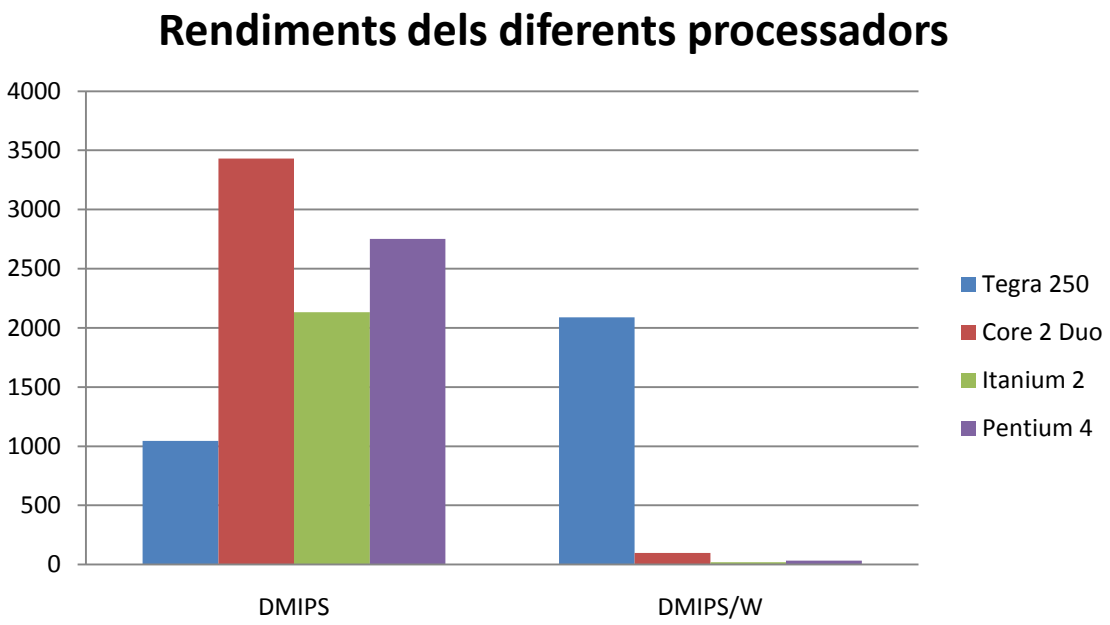

Figura 4.2: Resultats d'executar el Dhrystone benchmark

Els resultats són força favorables al Cortex A-9. Sent només 1,72 vegades inferior, (72% inferior), en DMIPS a la mitja del altres 3 processadors. És 41,23 vegades més eficient, (4023%), en rendiment DMIPS/W a la mitjana dels altres 3 processadors.

En quant al DMIPS declarats per ARM, són 4000 en front els 1045 que he obtingut jo corrent el benchmak. Com que ARM no especifica quins han estat els seus flags de compilació puc concluir que els meus resultats són més objectius per comparar amb altres processadors.

No obstant, veiem una dada que sorprèn. I és que un nucli de Core 2 Duo amb multithreding coarse grain i 2,1 GHZ tingui més DMPIS que un Pentiun 4 amb SMT i 3,2 GHZ. Això en aquest benchmark pot ser degut a la habilitat del Gcc per generar codi més eficient amb el flag -O3 pel Core 2 Duo que per al Pentium 4 però, també hem de tenir en compte que el Core 2 Duo va ser dissenyat per acabar més instruccions per cicle, (més grau en superescalavilitat), en comptes d'apostar per incrementar al màxim la freqüència com el Pentium 4.

4.5 Càlcul de coma flotant

4.5.1 Nbench benchmark

El Nbench és un benchmark estès que fa córrer una sèrie d'algorismes variats i molt usats en la computació. Està pensat per corre'l en un únic nucli, [4*].

A continuació explico que és cada algorisme:

- Ordenació numèrica: es correspon a ordenar una sèrie de nombres.
- Ordenació de strings: es correspon a ordenar una sèrie de cadenes de caràcters.
- Bitfield: algorisme que manega unes estructures de dades anomenades bitfields; força usat en l'àmbit de les ciències de la computació.
- Emulació de coma flotant: emula el potencial que tindrien les unitats de coma flotant.
- Fourier: algorisme de la transformada de Fourier; molt usat en computació d'altes prestacions.
- Assignació: es correspon amb un sèrie d'operacions d'assignació.
- Idea: (International Data Encryption Algorithm), algorisme internacional de xifrat de dades.
- Huffman: algorisme de codificació de Huffman.
- Xarxa neuronal: algorisme per simular xarxes neuronals.
- Descomposició LU: algorisme de descomposició low-uper de matrius; molt usat en computació d'alt rendiment.

4.5.2 Resultats i valoracions

	Cortex A-9 *	Cortex A-9 **	Core 2 Duo	Itanium 2	Pentium 4
Ordenació numèrica	533,6	440,91	893,6	867,84	859,52
Ordenació de strings	51,66	49,44	123,63	22,91	101,48
Bitfield	1,2979e+08	1,3111e+08	3,9757e+08	1,994e+08	5,0671e+08
Emulació de coma flotant	94,12	89,48	125,36	94,72	170
Fourier	369,48	382,25	22135	60233	19247
Assignació	5,79	5,63	25,69	18,651	31,31
Idea	912,34	986,19	3514,2	1533,6	2350,3
Huffman	481,51	652,17	2008,8	1427	1965,7
Xarxa neuronal	0,59	2,37	34,68	13,75	29,69
Factorització LU	18,312	225,84	1303,4	523,16	1144,9

Taula 4.5: Iteracions per segon de cada algorisme del nbench benchmark

Cortex-A9*: resultats amb compilació sense optimitzar per l'arquitectura.

Cortex-A9 **: resultats compilació optimitzada per l'arquitectura, usant les unitats hardware de coma flotant, (són els resultats agafats per fer la comparació).

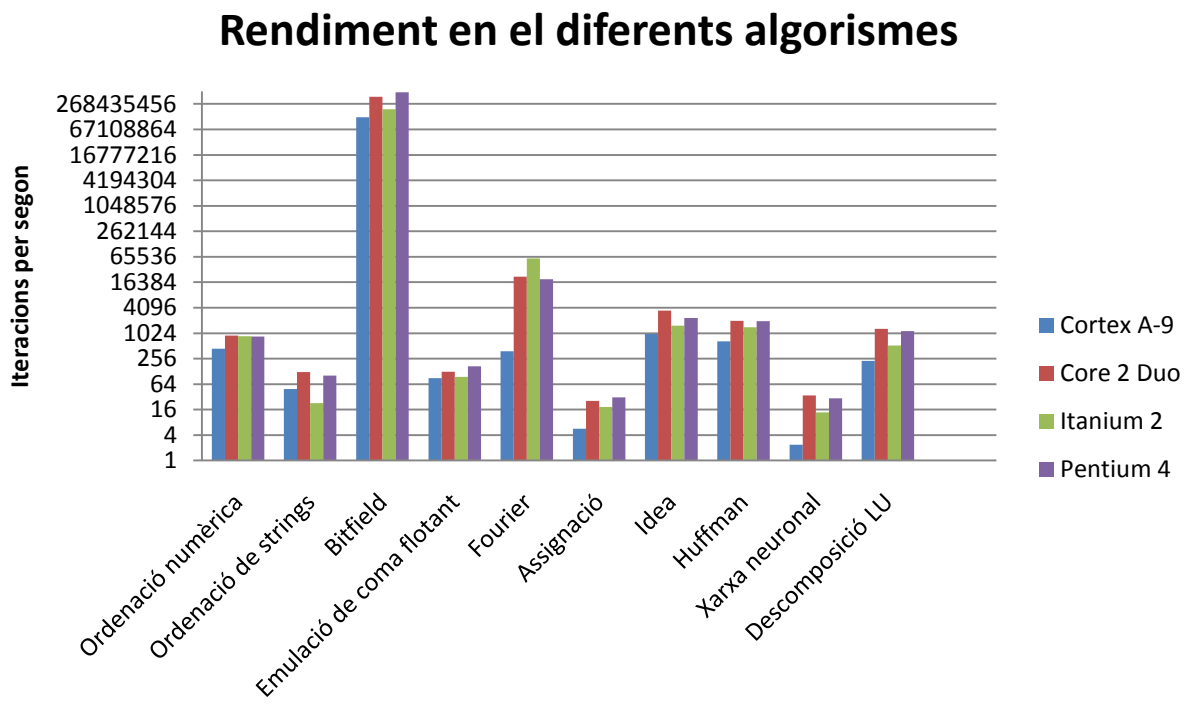

Figura 4.3: Iteracions/s en els diferents algorismes

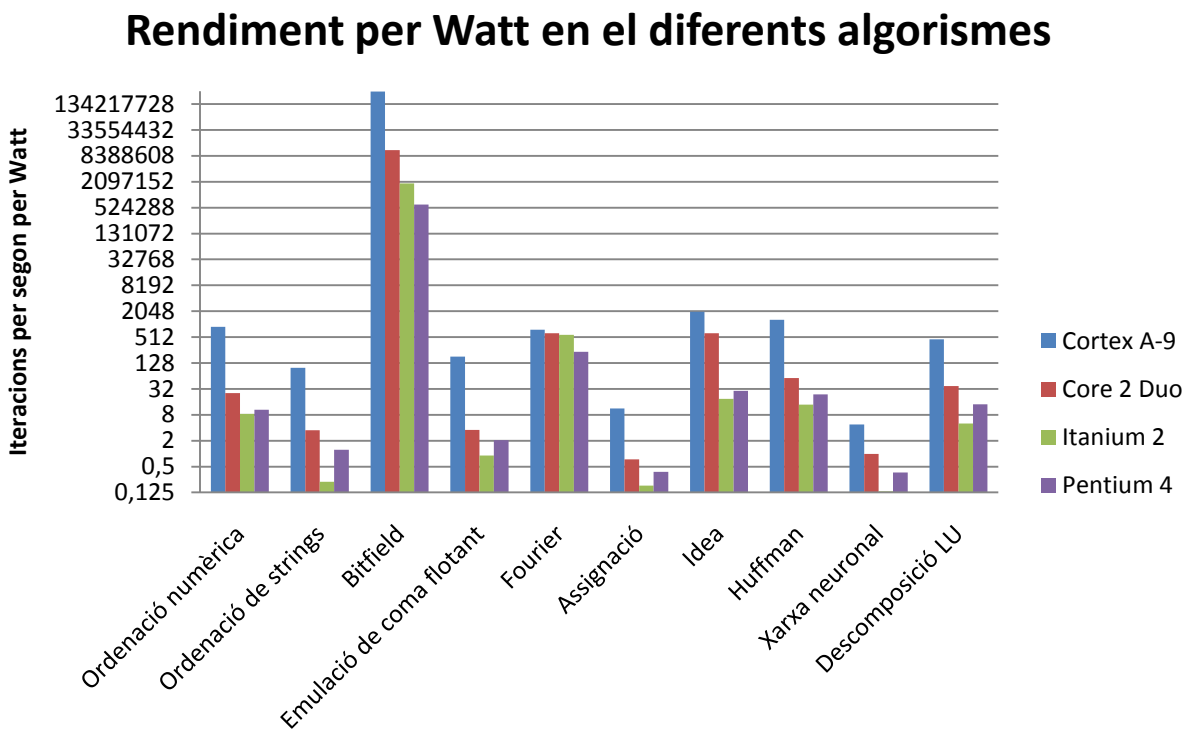

Figura 4.4: Iteracions/s per Watt consumit en els diferents algorismes

El CortexA-9 condicionat per estar dins del SoC Tegra 2 en la placa de desenvolupament, és 10,41 vegades més lent en promig que els altres 3 processdors. No obstant quan es tracta de rendiment per Watt, és 35,26 vegades més eficient en promig que els altres 3 processadors.

En alguns casos com en la transformada de Fourier o la xarxa neuronal el seu rendiment en comparació als altres tres processadors està molt per sota, sent 71 vegades més lent que la mitjana dels altres tres processadors en la transformada de Fourier i 10 vegades més lent en la xarxa neuronal.

Com ja hem vist en el rendiment per Watt el Cortex-A9 està al capdavant amb diferència; en el casos on treia pitjors resultats en rendiment iguala i supera en rendiment per Watt als altres tres processadors i en la resta de d'algorismes els supera de bon tros. Per exemple en la codificació de Huffman, el Bitfield i l'ordenació numèrica supera amb diferències de 23, 23 i 34 vegades més rendiment per Watt respecte el Core 2 Duo, (versió per ordinador portàtil), que és el segon processador que rendeix més per Watt.

Un dels algorismes més interessants per als objectius del projecte és la factorització LU, on amb aquest benchmak veiem com el Cortex-A9 rendeix 12 vegades més per Watt que el segon processador que rendeix més que és un Core2 Duo d'ordinador portàtil.

Si comparem el rendiment del Cortex-A9 amb el rendiment només del Core 2 Duo, tenim que per cada Core 2 Duo necsitariem 10 Cortex-A9 per igualar el rendiment. Si tinguéssim un blade amb 2 Core 2 Duo necessiteriem 20 Cortex-A9, la qual cosa implica un alt nivell de paral·lelisme que podria perjudicar significativament el rendiment del conjunt de processadors.

En definitiva, si el límit el posa el consum d'energia, el Cortex-A9 per si sol té un rendiment superior als demés processadors.

Capítol 5

5 Escalabilitat segons el nombre de threads

5.1 Com treballar amb threads

Per treballar amb threads hem de tenir instal·lada alguna llibreria de threads en el nostre sistema operatiu. En el meu cas he fet servir pthreads, que són una implementació molt usada. De fet, el compilador Gcc en la seva última versió ja porta tot el suport necessari per usar pthreads en els programes que fem. Només hem d'incloure la llibreria en la capçalera del programa i usar degudament les crides a la llibreria.

5.2 Creació i destrucció de threads

Per crear threads amb la llibreria de pthreads cal cridar a la funció pthread_create() i per destruir-los podem fer servir la funció pthread_join().

5.3 Producte escalar

5.3.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 3.

5.3.2 Mesures de temps i speed-ups

Les gràfiques següents es corresponen amb les mesures de temps i speed-ups de les execucions d'un producte escalar amb nombres de doble presició. En la primera gràfica l'eix del temps està a escala logarítmica en base 2. Les mesures del temps han estat preses fent la mitjana del temps d'usuari de 4 execucions consecutives corrent l'executable.

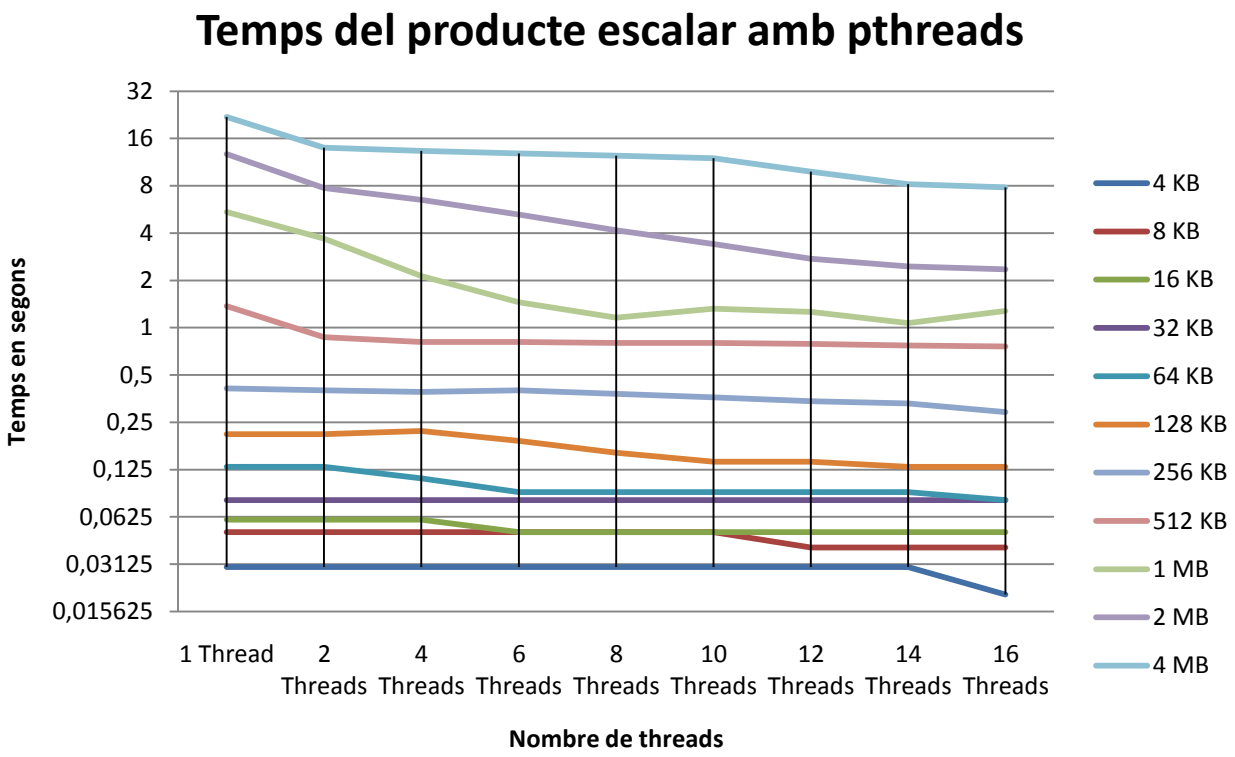

Figura 5.1: Gràfica dels temps del producte escalar amb pthreads

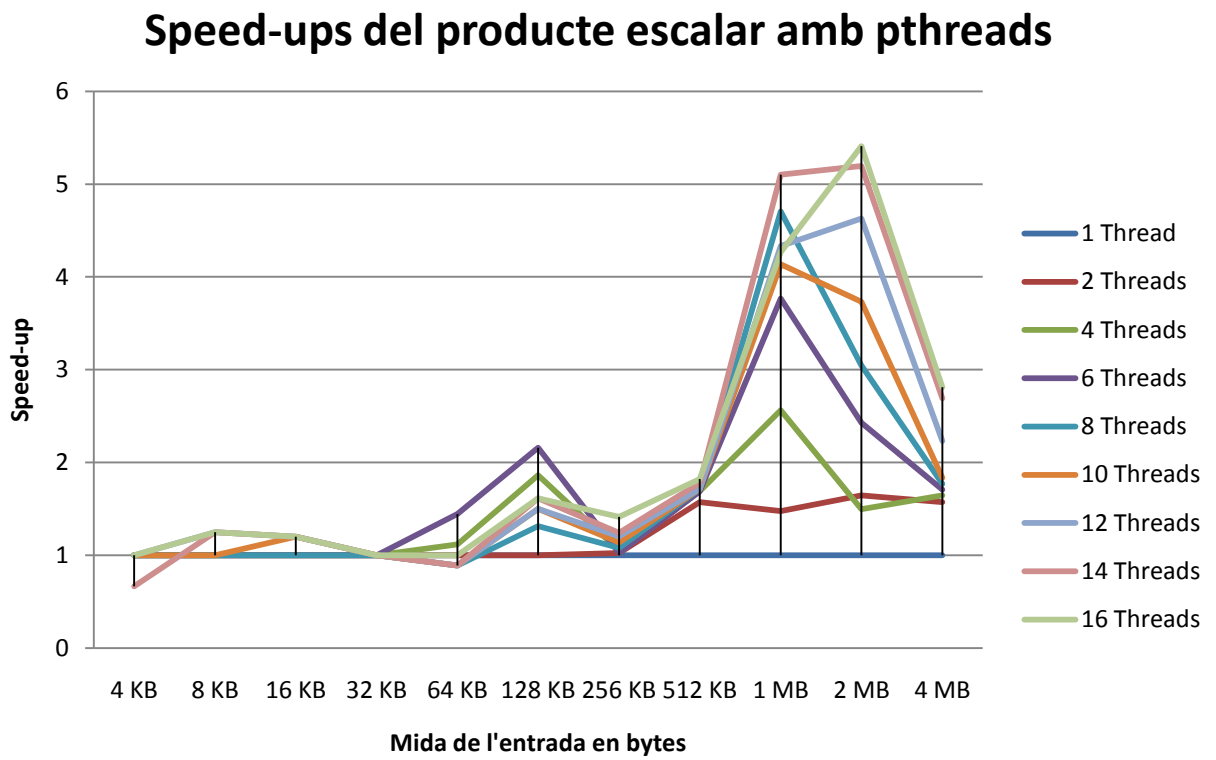

Figura 5.2: Gràfica dels speed-ups del producte escalar amb pthreads

5.3.3 Valoració dels resultats

El comportament del processador durant les execucions ha estat segons el que esperava. Les excucions en que els nuclis treballaven a 1 GHz els temps eren inferiors a les que alternava a 618MHz. No obstant això, la fiabilitat de les mesures de temps no s'ha vist gaire afectada, ja que cada grup de 4 execucions de cada nombre de threads ha estat executada sense pausa i amb pauses entre cada grup de nombre de threads. Per tant, tots els grups d'execucions han comneçat executant amb 1 GHz i han baixat a 618 MHz segons el conusum i la temperatura que s'analitzen en la secció mesures de consum i temperatura d'aquesta memòria. Pel que he observat al llarg d'aquest projecte, la freqüència no baixa apreciablement abans segons el nombre de threads sinó segons el rendiment que demana l'algorisme al processador.

Mentre que sense flags de compilació òptims, (temps que no he reportat en la memòria per no tenir utilitat pràctica), el programa no escala gens, amb flags de compilació òptims, com podem apreciar en la figura 5.2, si que escala.

En l'execució de 4MB amb 2 threads, compilant amb els flags òptims vaig obtenir un speed-up de 2,85 respecte la compilació sense flags òptims i un speed-up de 4,93 en l'execució de 4MB amb 16 threads respecte la compilació sense flags òptims.

A continuació quantifico exactament els speed-ups amb flags de compilació òptims:

Mida entrada	1 Threads	16 Threads	Speed-up
512 KB	1,37	0,76	1,80
1 MB	5,46	1,28	4,27
2 MB	12,73	2,35	5,42
4 MB	21,98	7,81	2,81

Taula 5.1: Mesures de temps i speed-ups assolits en funció del nombre de threads

En la gràfica de speed-ups veiem com a partir de 64 KB comença a escalar bé. Criden l'atenció les mides de 1 MB i 2 MB on s'aconsegueixen speed-ups de més de 5x. És a dir, que hi ha speed-up superlineal.

Resumint, podem dir que no obstant que per a les mides d'entrada més petites no escala segons el nombre de threads, ja que l'overhead de la gestió de threads es fa massa considerable respecte al temps d'execució total, quan incrementem la mida de l'entrada, a partir de 64 KB, l'algorisme escala.

5.4 Multiplicació de matrius

5.4.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 4.

5.4.2 Mesures de temps i speed-ups

Les gràfiques següents es corresponen amb les mesures de temps i speed-ups de les execucions d'una multiplicació de matrius amb nombres de simple presició. En la primera gràfica l'eix del temps està a escala logarítmica en base 2. Les mesures del temps han estat preses fent la mitjana del temps d'usuari de 4 execucions consecutives corrent l'executable.

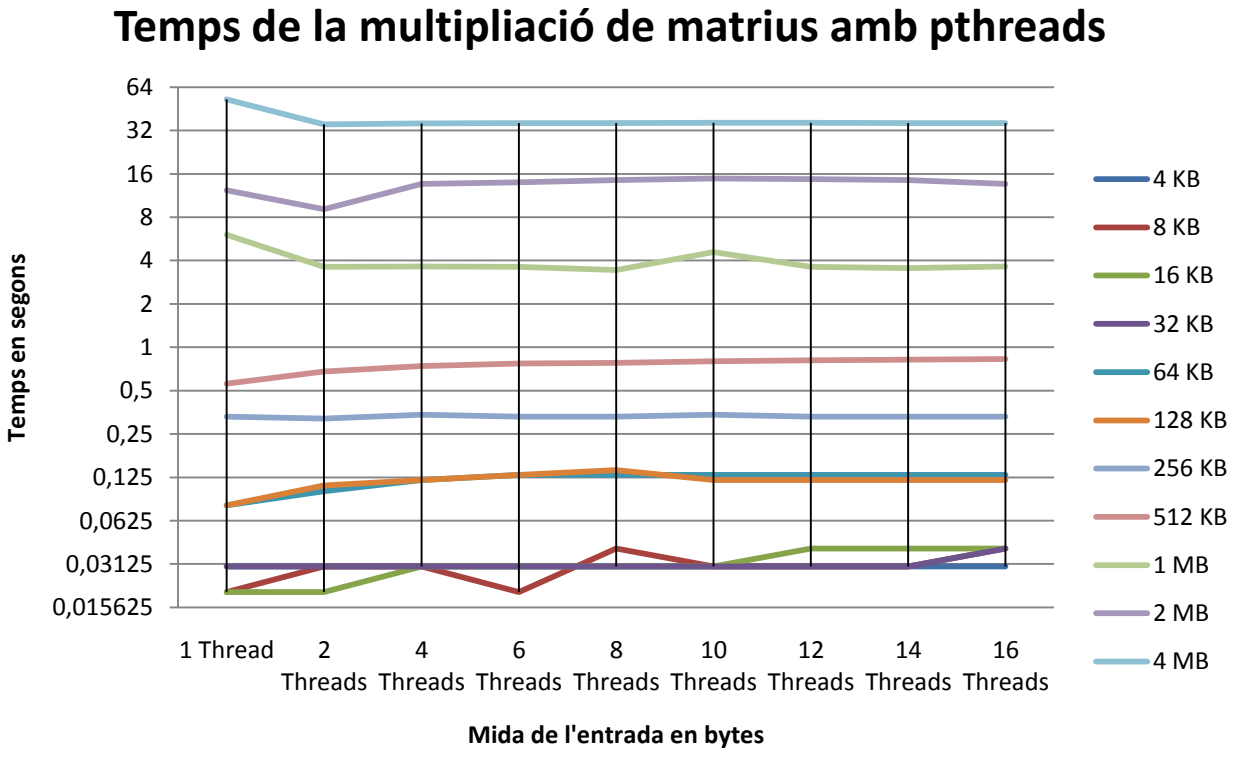

Figura 5.3: Gràfica dels temps de la multiplicació de matrius amb pthreads

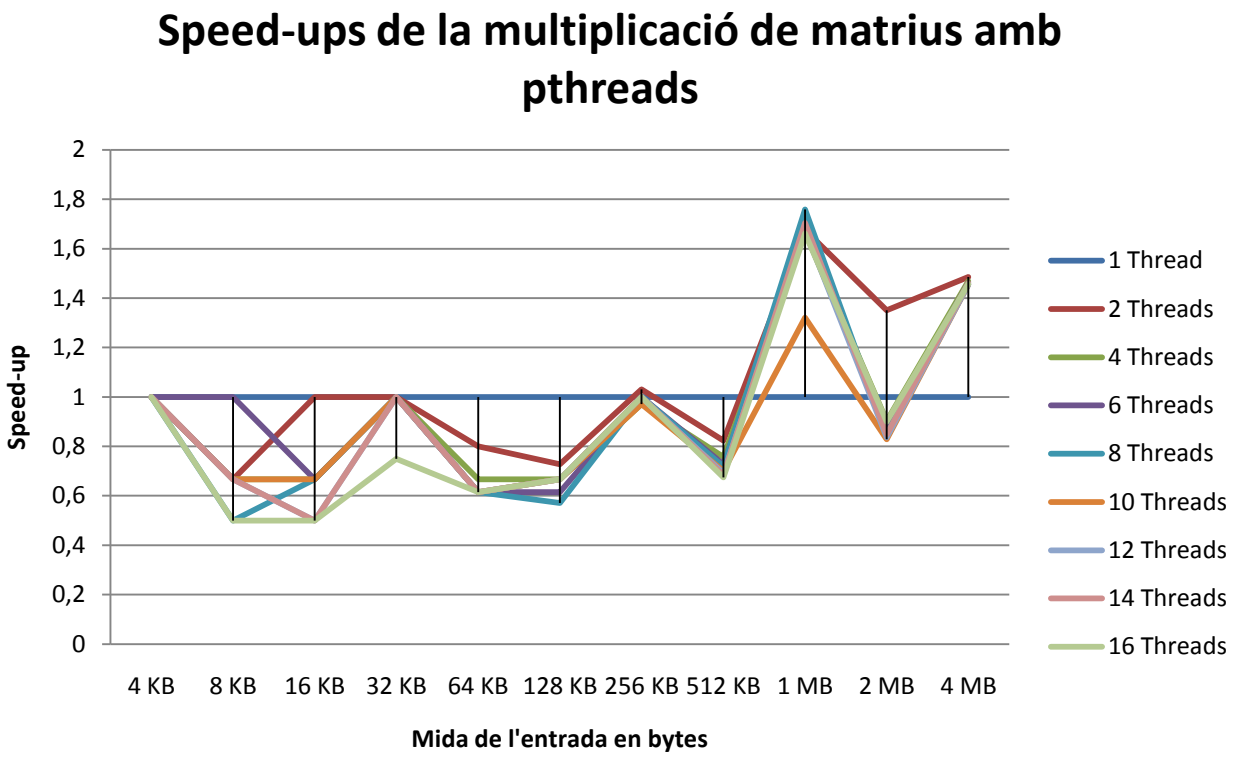

Figura 5.4: Gràfica dels speed-ups de la multiplicació de matrius amb pthreads

5.4.3 Valoració dels resultats

Com veiem en la figura 5.3, els temps d'execució amb més d'un thread no són molt més baixos que executant amb un sol thread, la qual cosa ens indica que no tenim un speed-up tant alt com en el cas del producte escalar. En aquest cas, en la figura 5.4 podem veure com no tenim speed-up fins a mides de 1 MB, a més a paritr de 1 MB casi no tenim speed-up a partir de dos threads. Per tant, aquest algorisme pràcticament no està escalant segons el nombre de threads sinó que el speed-up que estem obtenint és degut a l'ús del segon nucli.

Capítol 6

6 Integració d'OpenMP

6.1 Com treballar amb OpenMP

La configuració per treballar amb OpenMP en la placa és la mateixa que per a un PC normal. Només cal tenir instal·lada la última versió del compilador Gcc; en el meu cas ha estat la versió 4.3.3. A partir d'aquí, el que hem de fer al compilar és usar el flag –fopenmp, d'aquesta manera el compilador detectarà els "pragmas" d'OpenMP que haguem posat en el nostre codi.

Per configurar el nombre de threads i la planificació del scheduler, es fa segons les variables d'entorn ben conegudes. Només hem de fer "export OMP_NUM_THREADS=4", si per exemple volem executar el programa amb 4 threads. Per configurar la planificació es fa amb "export OMP_SCHEDULER="static"". En aquest cas he escollit planificació estàtica, que reparteix les iteracions a fer per cada thread a l'inici de l'execució, cal saber que aquesta planificació pot produir desbalanceig de càrrega segons la manera de recòrrer les estructures de dades.

6.2 Producte escalar

En aquest punt farem córrer exactament el mateix programa que en l'apartat 7.3 de la memòria no obstant, aquí farem servir OpenMP per comparar si la comoditat d'usar els pragmas d'OpenMP compensa alguna possible pèrdua de rendiment per paral·lelitzar automàticament amb OpenMP.

6.2.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 5.

6.2.2 Mesures de temps

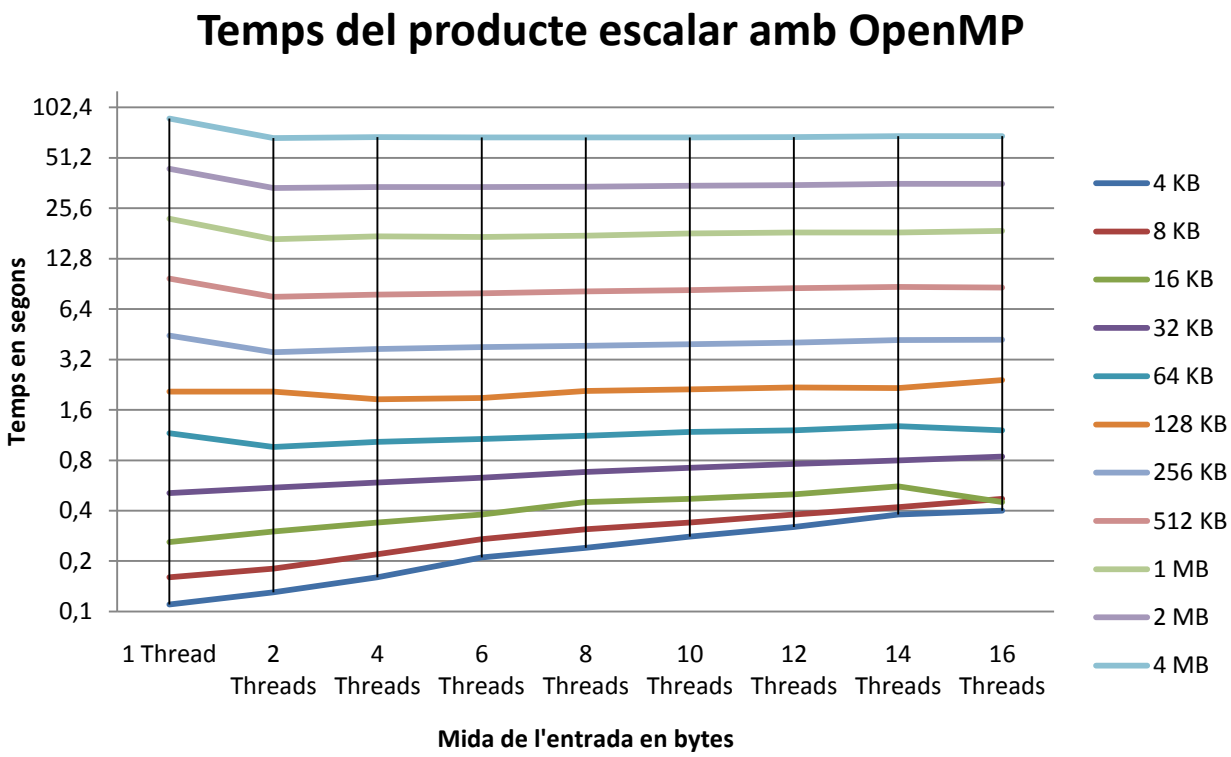

Figura 6.1: Gràfica dels temps del producte escalar amb OpenMP

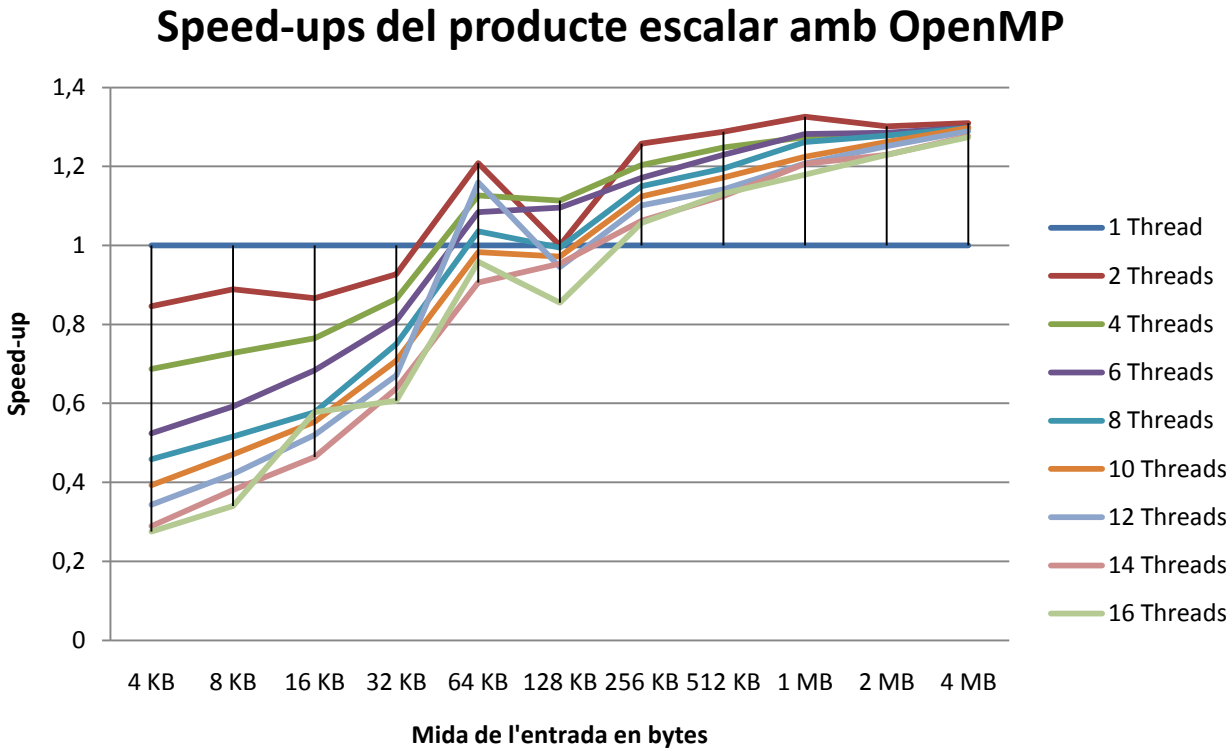

Figura 6.2: Gràfica dels speed-ups del producte escalar amb OpenMP

6.2.3 Valoració dels resultats

Per aquest algorísme, OpenMP té uns temps pitjors respecte a pthreads, usar OpenMP és casi igual de lent que si féssim un reserva de la variable suma a cada iteració, (aquest algorisme no està reportat a la memòria). Vegem un exemple de comparació de temps amb 1 thread i 512 KB a cada vector:

Algorisme	Temps
Producte escalar amb OpenMP.	15,78 s.
Producte escalar amb pthreads i reserva final.	6,96 s.
Producte escalar amb pthreads i reserva a cada iteració.	18,66 s.

Taula 6.1: Comparació de temps usant pthreads i usant OpenMP

El que ens porta a dir que en aquest cas OpenMP no està gestionant bé la reserva, de la variable.

Centrant-nos únicament en l'algorisme de les mesures, l'únic speed-up que obtenim en qualsevol cas és el d'usar el dos nuclis executant amb dos threads, a partir de més threads empitjoren els temps. El que és el mateix que dir que no hi escalabilitat segons el nombre de threads sinó que és degut a fer ús de més hardware.

6.3 Multiplicació de matrius

En aquest punt farem córrer exactament el mateix programa que en l'apartat de la memòria no obstant, aquí farem servir OpenMP per comparar si la comoditat d'usar els pragmas d'OpenMP compensa alguna possible pèrdua de rendiment.

6.3.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 6.

6.3.2 Mesures de temps

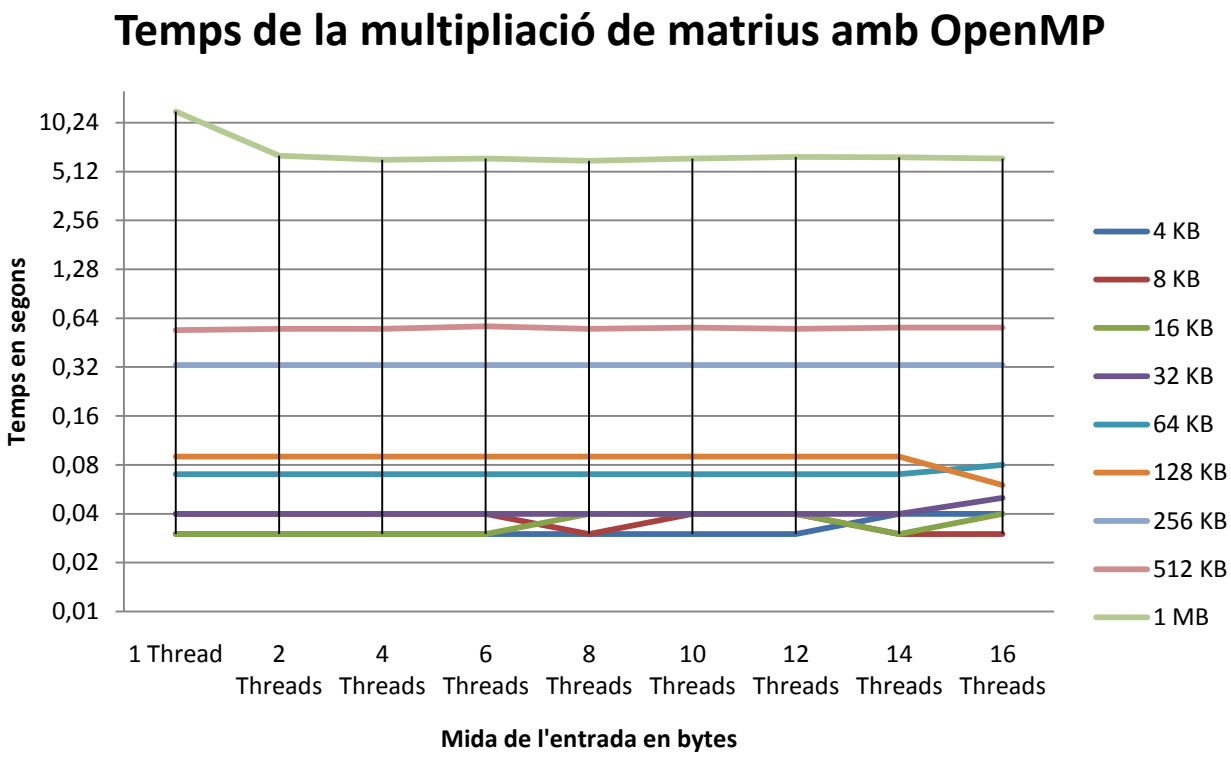

Figura 6.3: Gràfica dels temps del la multiplicació de matrius amb OpenMP

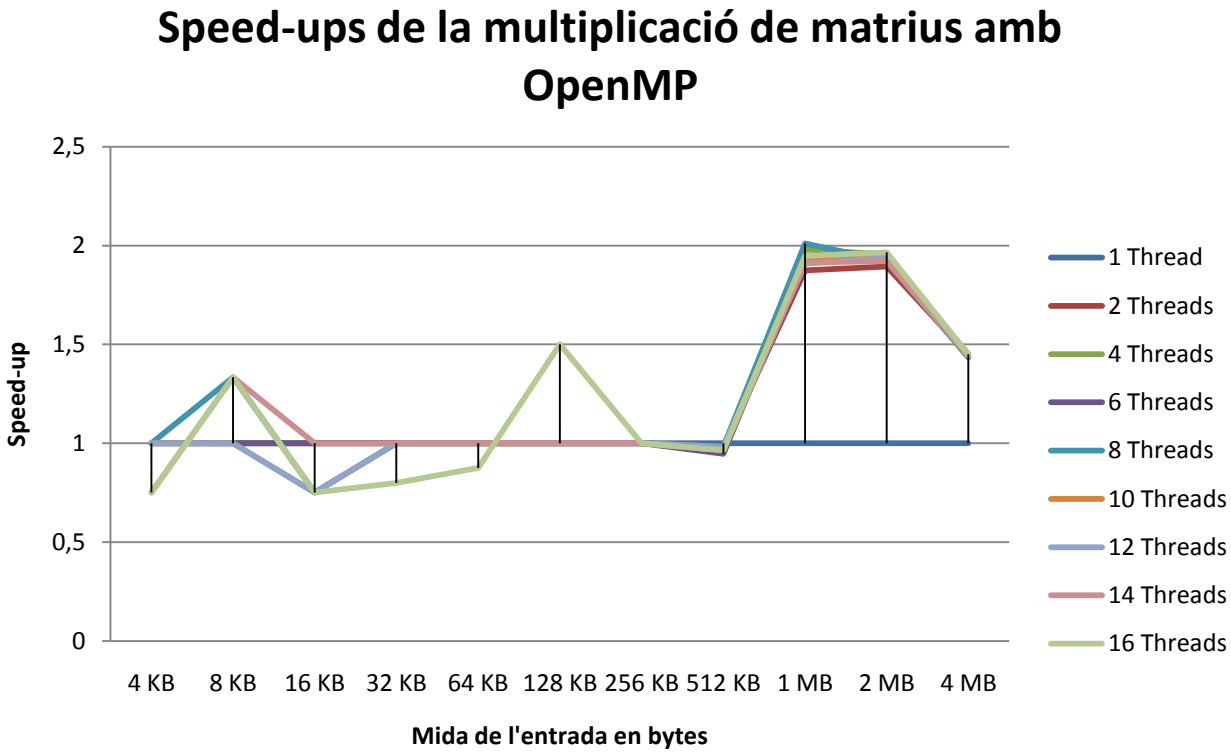

Figura 6.4: Gràfica dels speed-ups del la multiplicació de matrius amb OpenMP

6.3.3 Valoració dels resultats

En aquest algorisme he comprovat que els temps amb el rellotge inicialment en estat de repós no difereixen gens als temps amb el rellotge inicialment a 1 GHz, la qual cosa diu que el sistema de gestió de potència consumida del SoC dona tota la potència que cal amb un temps de resposta mínim.

També he comprovat que a cada execució el gestor pot decidir canviar de nucli on executar el programa; si les execucions són llargues és normal que ho faci per no escalfar massa un nucli en concret. No obstant, aquest fet pot perjudicar a la localitat de les dades en la memòria cau de nivell 1.

M'he asseguat de que al usar OpenMP amb dos o més threads, el rendiment dels nuclis és del 100% dins de les freqüències a a les que treballa, és a dir que no pel fet d'usar OpenMP es desaprofita capacitat de treball dels nuclis.

Per aquest algorisme, si comparem els speed-ups amb OpenMP respecte els de pthreads, tenim que amb OpenMP els speed-ups són en tots els casos millors que amb pthreads. Del que podem concloure que l'assignació d'elements de la matriu que fa OpenMP als threads està sent més eficient que la que s'està fent manualment amb pthreads. Per exemple, amb mida de 1 MB sense OpenMP, (amb pthreads), estem obtenint un speed-up de 1,8x mentre que amb OpenMP obtenim 2x.

En general, veiem com els temps s'incrementen més que en el producte escalar segons va augmentant la mida de l'entrada, això és degut a que la multiplicació de matrius té cost O(n^3) mentre que el producte escalar té cost O(n). Això també està influït perquè amb O(n^3) quan les dades no entrin en l'últim nivell de memòria cau haurem d'accedir fora molts més cops que en el producte escalar.

6.4 Factorització LU

En aquest punt he corregut una factoritazació LU usant tècniques de bloquing, paralel·litzant el treball tant a nivell interbloc com a nivell intrabloc.

6.4.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 7.

6.4.2 Mesures de temps

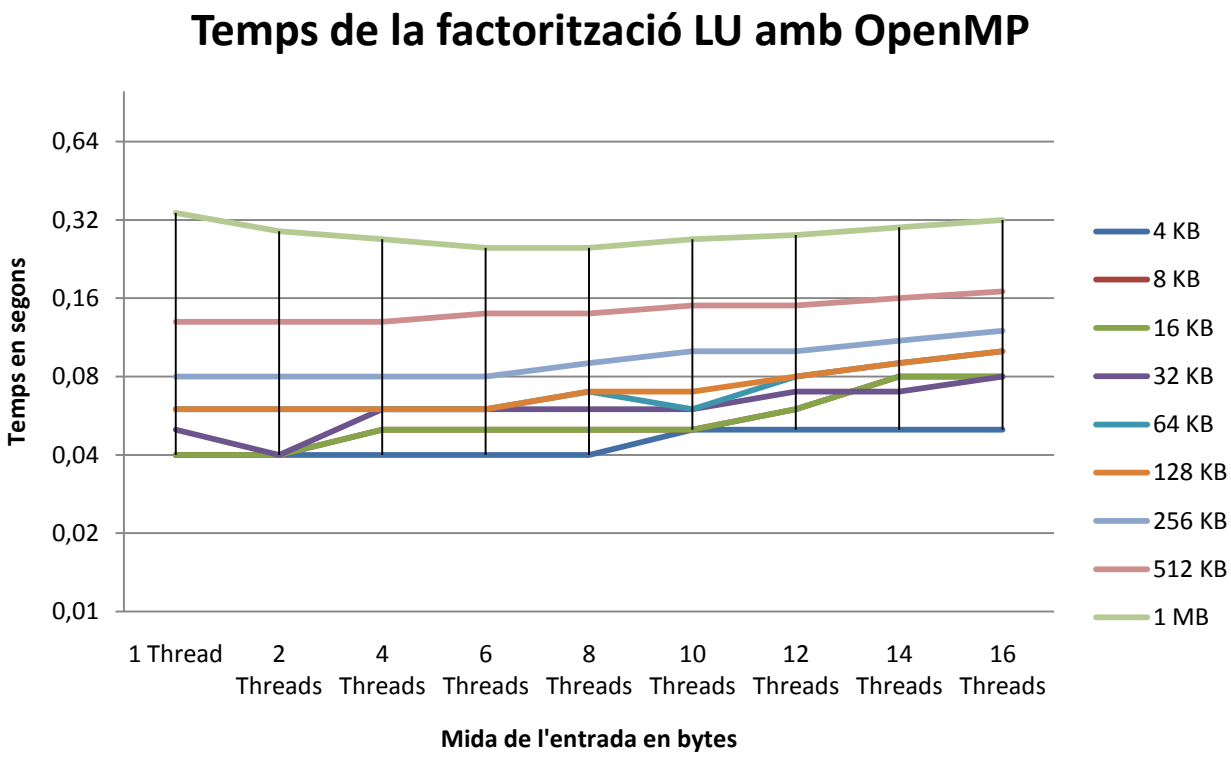

Figura 6.5: Gràfica dels temps del la factorització LU amb OpenMP

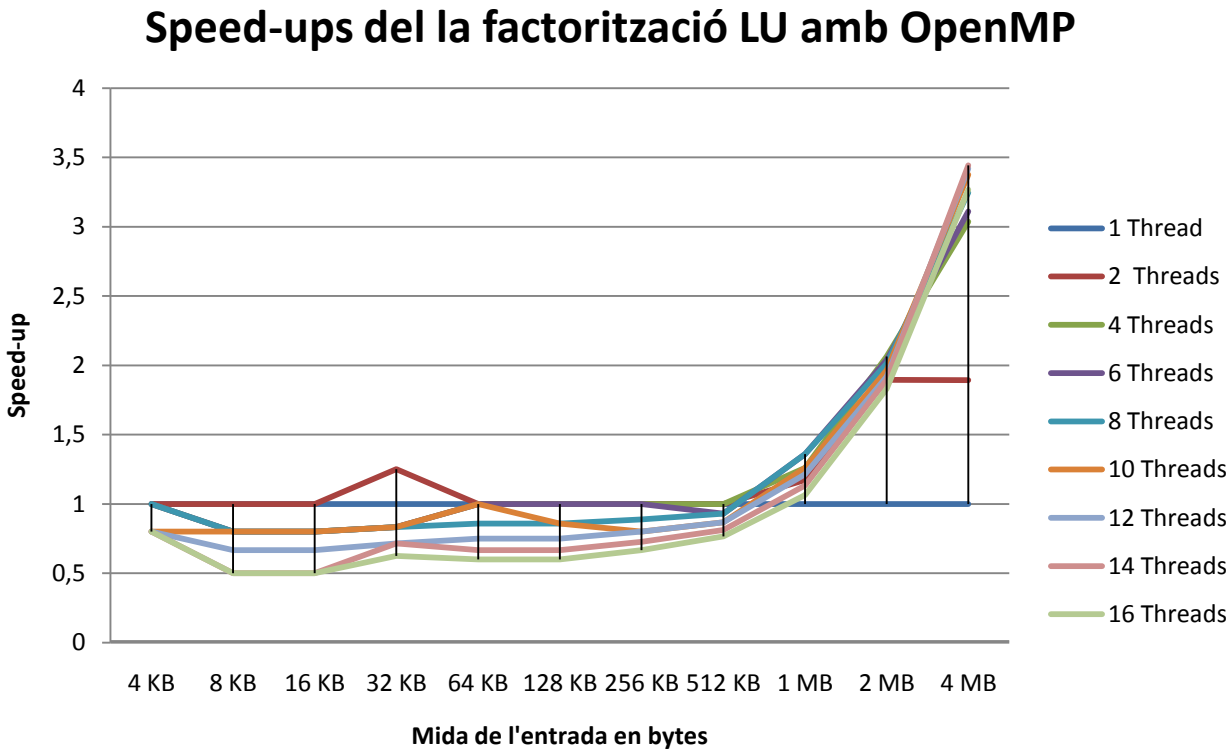

Figura 6.6: Gràfica dels speed-ups del la factorització LU amb OpenMP

6.4.3 Valoració dels resultats

En el cas del LU s'observa que escala bé no obstant, el nombre òptim de threads depèn de la mida de l'entrada. Per la mida de 4 MB el nombre de threads òptim és14 mentre que per la mida de 2 MB el nombre òptim és 4 threads. Per a la resta de les entrades el nombre òptim està entre 4 i 6 threads.

Com a speed-ups significatius en el Cortex-A9 tenim, per a la mida d'entrada de 4 MB, 3,44 en l'execució de 14 threads respecte la de 1 thread, i comparant execucions on s'utilitzen els dos nuclis 1,81 en l'execució de 14 threads respecte la de 2 threads. És a dir, que esta escalant segons el nombre de threads i no només degut a l'ús de més hadware.

A continuació comparo els rendiments i rendiments per Watt del Cortex-A9 amb els del Pentium4, per a les mides d'entrada de 2 MB i 4MB.

	Millor temps Cortex-A9	Millor Temps Pentium 4	Aventatge Rendiment P4	Aventatge Rendiment per Watt Cortex-A9
2 MB	0,79 (4 threads)	0,14 (8 threads)	5,64x	29,19x
4 MB	1,49 (14 threads)	0,35 (6 threads)	4,26x	39,41x

Taula 6.2: Comparació de LU amb Cortex-A9 i Pentium 4

6.5 Factorització Cholesky

En aquest punt he corregut una factorització Cholesky amb algorisme seqüencial sense blocking i paral·lelitzat amb OpenMP.

6.5.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 8.

6.5.2 Mesures de temps

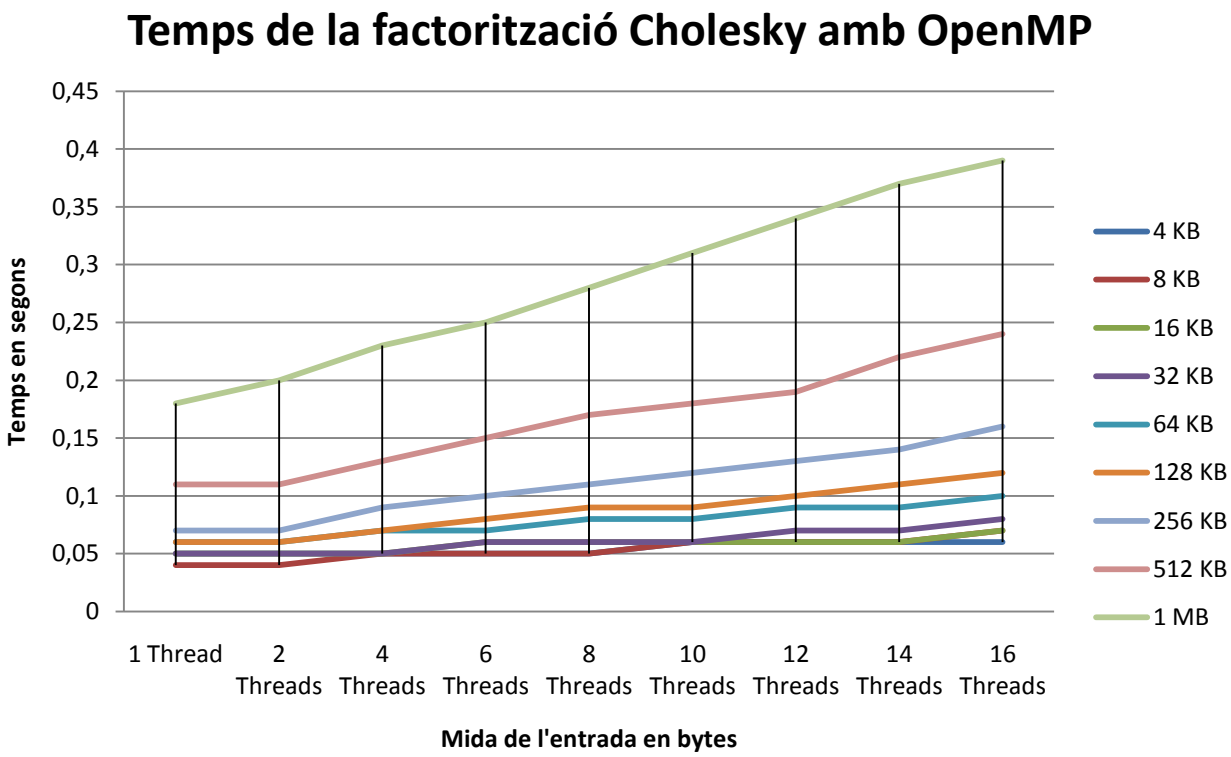

Figura 6.7: Gràfica dels temps del la factorització Cholesky amb OpenMP

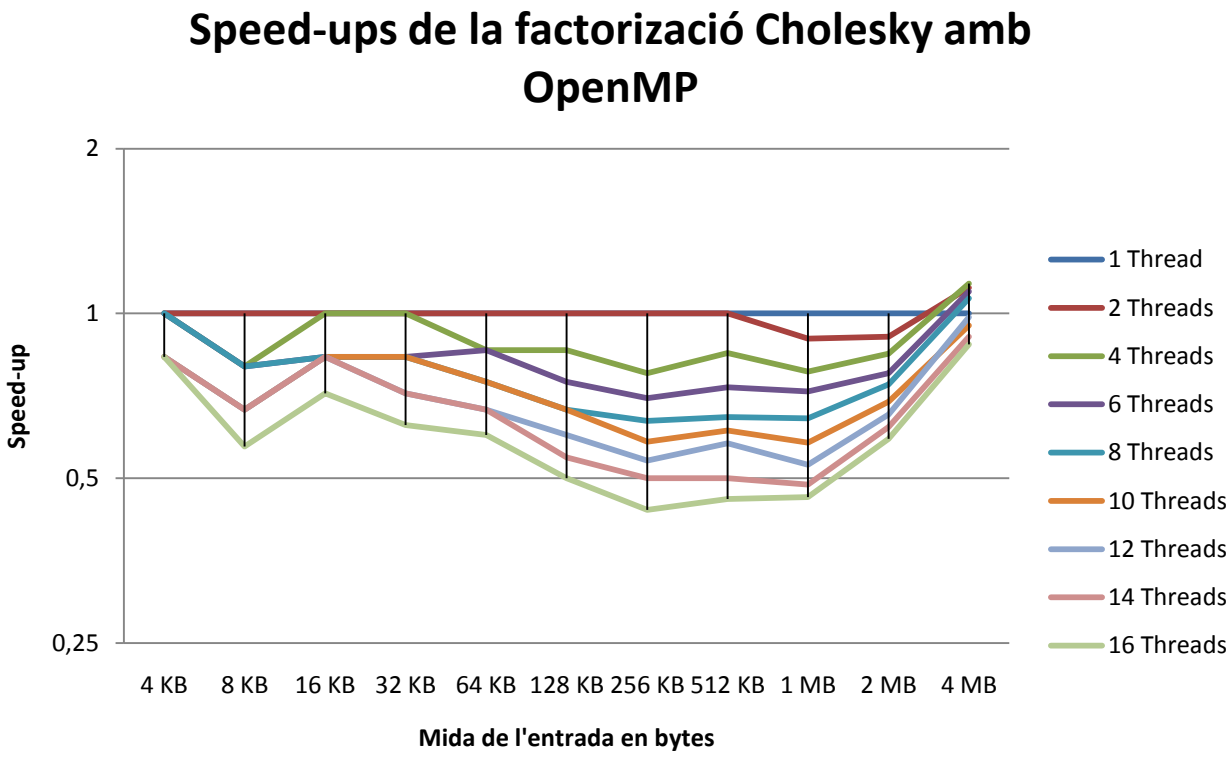

Figura 6.8: Gràfica dels speed-ups del la factorització Cholesky amb OpenMP

6.5.3 Valoració dels resultats

Podem apreciar com els temps són més del doble de ràpids que en la factorització LU això és degut a que el Cholesky només tracta la meitat diagonal superior de la matriu i per tant opera aproximadament amb la meitat de la matriu.

En aquest algorisme veiem com no hi ha escalabilitat segons el nombre de threads, tan sols en l'execució per a la mida d'entrada més grossa tenim el millor rendiment amb 4 threads; a partir de més threads el rendiment va caient degut a overhead de gestió dels threads. Realment en aquest punt estem veient com OpenMP amb planificació estàtica no es capaç d'escalar gaire segons el nombre de threads. Per aconseguir speed-ups en factoritzacions haurem de fer ús de tècniques de bloquing per aprofitar la localitat temporal i de ben segur canviar la planificació per no tenir desbalanceig de càrrega.

Capítol 7

7 Integració de MPI

Per poder computar amb molts processadors cal usar varis nodes, on entenem node com a cada conjunt de processadors que comparteixen memòria física. Per que tots el nodes puguin computar com un conjunt cal una interfície que permeti el pas d'informació entre tots els nodes. Actualment la interfície més usada és MPI, interfície de pas de missatges.

7.1 Muntatge del clúster Beowulf

S'anomena clúster al conjunt de computadores que s'uneixen amb l'objectiu de que es comportin com si fossin una amb més prestacions. Podem distingir varis tipus de clústers entre ells.

- Beowulf: els clústers Beowulf serveixen per aconseguir molts FLOPS i computar més ràpid els algorismes. Prenen el seu nom d'un projecte de la NASA del 1994 que va consistir en connectar 486 processadors sota una xarxa Ethernet amb l'objectiu d'aconseguir alt rendiment paral·lelitzant les aplicacions usant interfícies PVM i MPI.
- SSI: single-system image; són extensions per a Linux que permeten convertir un conjunt de màquines en un clúster, és el que s'anomena SSI. Els SSI tenen l'objectiu de crear un clúster de manera molt més fàcil d'administrar i d'usar. Exemples de SSIs són:
    - OpenMosix: A OpenMosix no cal paral·lelitzar les aplicacions ja que ell mateix s'encarrega de mapejar els processos al diferents nodes, (migració de processos). No suporta el mapeig de fils d'execució i s'utilitza majoritàriament per paral·lelitzar aplicacions que tenen molta entrada/sortida (I/O).
    - Kerrighed: també és un extesió per a Linux com OpenMosix amb la diferència de que si pot mapejar threads però, si necessita que els codis estiguin paral·lelitzats. Tant OpenMosix com Kerrighed són sofware que permeten convertir un conjunt de màquines en un clúster, [5].
- HA amb LVS: high avaliavility amb balancejador de càrrega linux virtual server. Usats principalment en granges de servidors. Es componen de les màquines reals on tenim corrent Apache, el balancejador de càrrega composat d'un balancejador

mestre i un backup que reparteixen les peticions entre els diferents servidors reals i les dades a servir. Alguns projectes de HA són:

- Ultramonkey: és un projecte que simplifica la configuració del clúster HA.
- LVS + Keepalived: també és un projecte com ultramonkey que utilitza altres eines per monitorizar el balancejador de càrrega i els servidors.

7.1.1 Esquemes de les xarxes

Pel muntatge de la xarxa, com que per fer les proves només tenia dues plaques vaig optar per usar un router sense necessitat d'utilitzar un switch.

Primer de tot vaig comprovar quin rendiment tenia un processador com és ara un Peintium 4 amb l'ajut de dos Cortex-A9 que tenen un potencial de càlcul molt inferior, fent així un clúster heterogeni.

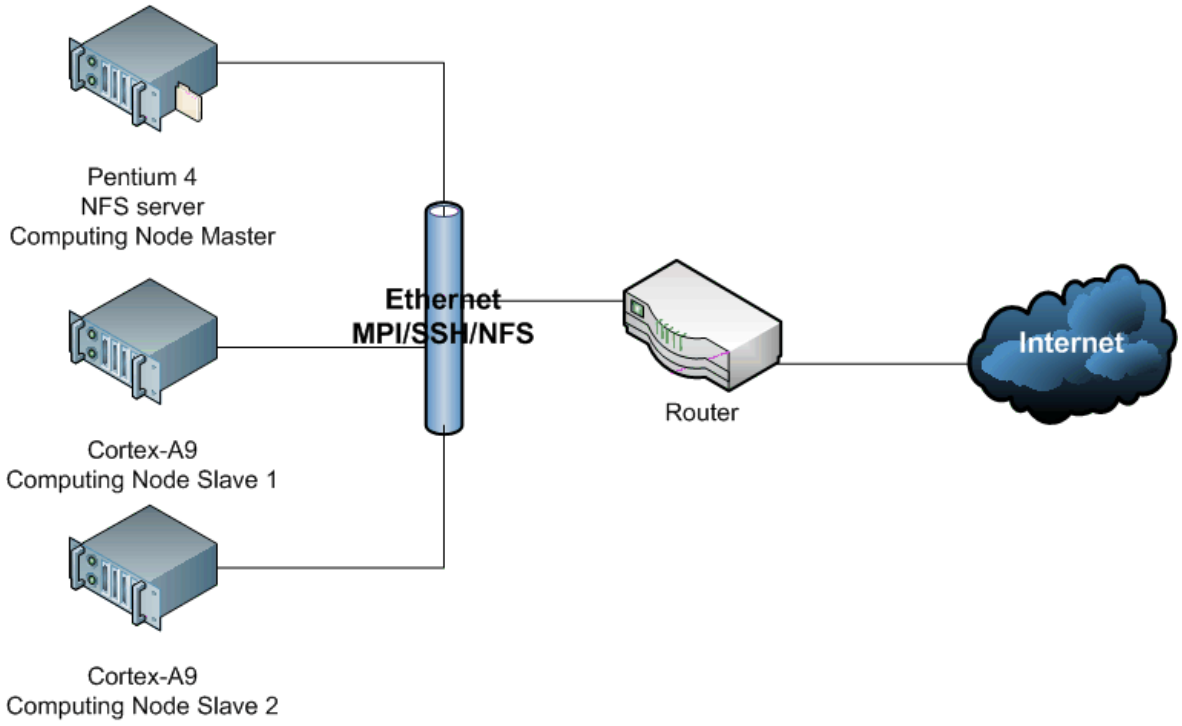

Figura 7.1: Esquema del clúster heterogeni

El resultat va ser que no es guanyava gaire amb la cooperació ja que la proporció guanyada en cooperació de còmput no era massa superior a l'overhead de les comunicacions amb una xarxa Ehternet de 100 Mbps.

Com a segona configuració, la que he usat per fer totes les proves, vaig configurar una placa com a mestre i l'altre com a esclava. Deixant al node del Pentium 4 només com a servidor NFS, per facilitat de treball ja que és on tenia l'entorn gràfic.

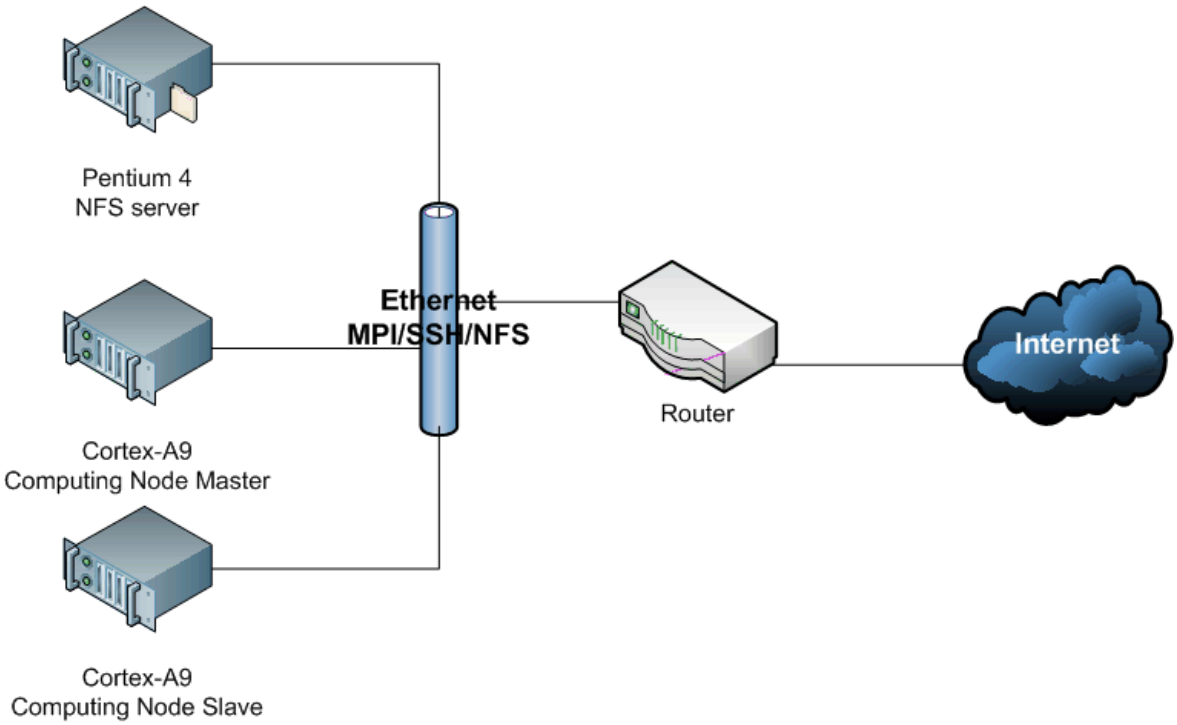

Figura 7.2: Esquema del clúster homogeni

En aquest segon cas si que es pot parlar de bona escalabilitat per a dues plaques, amb speed-ups al voltant del lineal, que en alguns casos són superlineals.

7.1.1.1 Els nodes del clúster

Figura 7.3: Fotografia de les plaques que conformen el clúster

7.1.2 Configuració de NFS

7.1.2.1 Creació usuari

- sudo useradd -p clusty -s /bin/bash -d /home/clusty clusty
- sudo passwd clusty
- sudo mkdir clusty
- sudo chown clusty:clusty /home/clusty –R
- [23*], [24*], [30*].

- ls -la
    - total 16
    - drwxr-xr-x 4 root   root   4096 2010-08-27 13:35 .
    - drwxr-xr-x 25 root   root   4096 2010-08-27 13:30 ..
    - drwxr-xr-x 2 clusty clusty 4096 2010-08-27 13:35 clusty
    - drwxr-xr-x 53 david david 4096 2010-08-27 13:30 david

- su clusty
- echo $HOME
    - /home/clusty

- sudo usermod -G admin clusty (des d'un usuari amb permisos d'administració).
    - Aquest últim pas és per comoditat alhora de treballar amb l'usuari però, suposa un forat de seguretat.

7.1.2.2 Configuració comú del servidor i dels clients

- /etc/hosts: serveix per fer l'associació entre direccions ip i noms de hosts, perquè sigui més còmode editar fitxers, hem d'afegir tots els clients del nostre clúster:
    - 192.168.1.65 david-desktop
    - 192.168.1.66 tegra-ubuntu
    - On primera columna és la direcció IP assignada al equip, i la segona columna és el nom del equip.
    - Un cop fet això hem de reiniciar la xarxa amb, sudo /etc/init.d/networking restart
    - Recordem que podem canviar el nom de l'equip a /etc/hostname

7.1.2.3 Configuració servidor

- Instal·lar els paquets NFS per al servidor, [37*], [38*], [42*]:
  - sudo apt-get install nfs-kernel-server nfs-common portmap

- /etc/exports: aquest és el fitxer que indica quins directoris volem exportar com a servidor, com volem exportar-los i a quins clients volem exportar. Per exemple:
  - /home/clusty 192.168.1.0/255.255.255.0(rw,no_root_squash).

- Si els volguéssim exportar més còmodament amb l'ajut del fitxer /etc/hosts ho faríem així:
  - /home/clusty tegra-ubuntu(rw, no_root_squash)
  - Recordem que el nom del equip, (host), a ubuntu es troba al fitxer /etc/hostname.

- En qualsevol cas perquè els canvis del fitxer /etc/exports tinguin efecte hem de fer:
  - sudo exportfs –a

- Per comprovar els directoris muntats pel servidor podem fer: (on la IP del exemple és la IP que tingui el servidor).
  - showmount –e 192.168.1.65

7.1.2.4 Configuració clients

- Instal·lar els paquets NFS pel client:
  - sudo apt-get install nfs-common portmap

- Muntar el directori exportat pel servidor en un directori en el client, això ho podem fer així:
  - sudo mount –t nfs 192.168.1.65:/home/clusty /home/clusty
  - On la IP és la IP del servidor i el path de la ultima columna és el path al client, un path que ha d'haver estat creat abans.

- O si volem que es faci automàticament al iniciar el sistema podem editar el fitxer /etc/fstab i afegir la següent entrada:
  - 192.168.1.65:/home:clusty /home/clusty nfs defaults 0 0

- I fer: sudo mount –a; per muntar-ho durant aquesta sessió sense haver de reiniciar.

7.1.3 Configuració de ssh

- Si no el tenim, instal·lem el openssh-server al node esclau, [40*]:
  - sudo apt-get install openssh-server

- Des del node mestre:
  - ssh-keygen –t rsa –f ~/.ssh/id_rsa
  - eval ‘ssh-agent –s’
  - ssh-add
  - en el punt ssh-add podem trobar amb que necessitem fer abans: exec ssh-agent bash
  - scp ~/.ssh/id_rsa.pub clusty@tegra-ubuntu:/~/.ssh/id_rsa.pub
  - on ~ és el directori personal de l’usuari, en aquest cas podia haver estat reemplaçat per /home/clusty.

- Ara a tots els nodes esclaus
  - cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

- Node mestre, com que ara tenim NFS amb el directori personal importat no ens funcionarà el mètode, per fer-ho funcionar hem de crear un directori en el node mestre amb els fitxers tal com hem fet en el node esclau.
  - mkdir .ssh
  - cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
  - ssh tegra-ubuntu

- Node esclau: reiniciar ssh
  - sudo /etc/init.d/ssh restart

- Un possible problema pot ser que en el fitxer /etc/ssh/sshd_config no estigui ben configurat, ens hem d’assegurar dels següents paràmetres: on AuthorizedKeysFile pot estar com alguna de les tres maneres:
  - RSAAuthentication yes
  - AuthorizedKeysFile %h/.ssh/authorizedkeys

  - #AuthorizedKeysFile %h/.ssh/authorizedkeys
  - AuthorizedKeysFile /home/clusty/.ssh/authorized_keys

- Una altre possible problema és que si esteu usant un servidor DHCP per a assignar IPs al clúster algun dia us canviïn les IPs als nodes. En tal cas s'han de canviar els arxius /etc/hosts, a més al connectar per ssh amb MPI us demanarà constantment comprovacions per fer les connexions ssh, us sortirà un missatge amb el format; "Warning: the RSA host key for X differs from the key for the IP address Y" on X és el nom del host a connectar i la IP és la IP antiga que tenia, això és una protecció per evitar un possible atac de DNS spoofing. Per arreglar-ho cal a anar al fitxer /etc/ssh/ssh_config i editar la següent línia:
  - CheckHostIP no
  - I fer un: sudo /etc/init.d/sshd restart

7.1.4 Configuració amb MPICH 2

Com que MPICH1 donava problemes de mapeig de processos MPI als diferents nuclis vaig decidir usar MPICH2. És per això que els apartats d'instal·lació i ús de MPICH1 han estat omesos en la memòria, [25*], [39*], [43*], [47*].

MPICH2 està disponible a partir de les distribucions Karmic i Lucid, (9.10 i 10.04). Per la qual cosa amb la distribució utilitzada, Jaunty, (9,04), no tenim als repositoris MPICH2. Per poder accedir als repositoris de Lucid farem el següent a cada node del clúster:

- Primer de tot si abans teníem instal·lat MPICH1, desinstal·larem tots els seus paquets amb:
  - sudo apt-get –purge remove "paquets de MPICH1".

- Després substituïm els repositoris on cercar els paquets.
  - sudo vim /etc/apt/souces.list .

- Substituïm Jaunty per Lucid.
  - sudo apt-get install mpich2.

7.1.5 Compilar i executar amb MPICH 2

El cas de que estem en un clúster heterogeni no té utilitat pràctica en aquest projecte i les pases per compilar i executar han estat omeses. A continuació mostro les passes per compilar i executar en un clúster homogeni, [44*], [45*].

La compilació en MPICH2 funciona amb els mateixos paràmetres que amb MPICH1. No obstant, cal recordar que els executables compilats amb l'eina mpicc de MPICH1 no són compatibles amb MPICH2.

- Per compilar hem d'usar la comanda mpicc on X és el nom del nostre programa
    - mpicc –o X X.c
- Per executar primer hem de aixecar el clúster amb la comanda mpdboot, on –n indica el nombre de nodes que volem i –f apunta el camí on es troba el fitxer on estan especificats els nodes del clúster.
    - Mpdboot –n 2 –f /home/clusty/mpd.hosts
- Per executar cal fer la comanda mpiexec, on n indica el nombre de processos MPI que volem; en la pròpia comanda ja es fa automàticament el mapeig de processos MPI als processadors dels nodes.
    - Mpiexec –n 2 ./hytest.c

7.2 Factorització LU

7.2.1 Algorisme Paral·lelitzat

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 9.

7.2.2 Dependències de dades

7.2.2.1 Algorisme seqüencial

```
for (k=0; k<N; k++){
     for(i=0; i<N; i++){
          A[i][k] = A[i][k] / A[k][k];
       }
       for(i=k+1;i<N;i++){
             for(j=k+1;j<N;j++){
              A[i][j] = A[i][j] - A[i][k] * A[k][j];
             }
       }
}
```

Codi 7.1: Algorisme de factorització LU sequencial

Iteració 1: k=0

Figura 7.4: Iteració 1 factorització LU serial

Iteració 2: k=1

Figura 7.5: Iteració 2 factorització LU serial

Com veiem primer s'han de calcular les divisions amb elements de la primera columna començant per l'element de la fila iteració + 1 respecte de l'element de la cantonada superior dreta, per posteriorment calcular el mòdul de tots els elements de les files i columnes a partir de la iteració + 1 a partir de la primera columna i la primera fila respectivament.

7.2.2.2 Algorisme per blocs

En l'algorisme per blocs per MPI operem a dos nivells; els blocs en que es divideix la matriu, els quals s'intercanviaran per missatges MPI entre els nodes, i els elements dins del bloc, el quals podem repartir el seu còmput mitjançant threads; per tant paral·lelitzem a dos nivells, el nivell de blocs és el que es diu paralel·lització de gra gros i el nivell dins del bloc que es el que es diu paralel·lització de gra fi.

Podem fer varies estratègies, la més comú és fer una mida de bloc fixa i assignar a cada processador una fila de blocs de la matriu fins que no tenim més processadors per files i tornem a començar l'assignació des del primer processador. Una altre opció és fer la mida dels blocs variable de manera que la mida del bloc sempre permet aprofitar tots els processadors que tinguem i evitar la inanició fins que tinguem una matriu prou petita com per fer blocs de mida fixe. En el meu cas, tenint dos processadors, un en cada node, no té gaire sentit fer blocs de mida variable ja que els blocs a transmetre de un node a un altre serien massa grossos al principi a més de que només s'enviarien 2 blocs que trigarien massa temps a arribar fins que el receptor pogués començar a calcular. Vegem un exemple de com serien les dues primeres iteracions amb mida de bloc fixe. On al processador 1 li tocarien les linies imparelles i al processador 2 les parelles.

Iteració 1:

LU0	FWD	FWD	FWD	FWD
BDIV	BMOD	BMOD	BMOD	BMOD
BDIV	BMOD	BMOD	BMOD	BMOD
BDIV	BMOD	BMOD	BMOD	BMOD
BDIV	BMOD	BMOD	BMOD	BMOD

Figura 7.6: Iteració 1 factorització LU amb bloquing

Iteració 2:

	LU0	FWD	FWD	FWD
	BDIV	BMOD	BMOD	BMOD
	BDIV	BMOD	BMOD	BMOD
	BDIV	BMOD	BMOD	BMOD

Figura 7.7: Iteració 2 factorització LU amb bloquing

L'ordre de tractament dels blocs ha de respectar les dependències de dades. Primer un processador, (o node), ha de calcular el bloc LU0, el qual s'haurà d'enviar a tots els altres processadors, ja que els blocs BDIV necessiten les dades del bloc LU0. Després per cada FWD que faci el processador que li toca la primera fila l'haurà d'enviar a tots els altres processadors perquè puguin calcular el seu bloc BMOD respecte de la columna del bloc FWD enviat.
Respecte al tractament de cada tipus de bloc, s'han de mantenir les operacions que fa l'algorisme seqüencial. Per això en el bloc LU0 es fa un algorisme LU seqüencial, en el bloc BDIV es fa també un algorisme LU seqüencial i en els blocs FWD i BMOD es fa un algorisme LU seqüencial només amb la part dels mòduls dels elements, en el cas del FWD no cal fer els mòduls de la primera fila del bloc.

7.2.3 Mesures de temps i gràfiques

Per a les mesures de temps amb MPI he variat el nombre de processos MPI i el nombre de threads OpenMP per a determinades mides de dades d'entrada.

MPICH2 ha mapejat els processos MPI de manera que els processos imparells s'allotjen en el processador de la placa màster i els processos parells s'allotjen en la placa esclava.

En les següents taules la primera columna indica els threads OpenMP mentre que la primera fila indica els processos MPI.

Enlloc de prendre les mesures amb un procès MPI he optat per córrer el codi sèrie amb OpenMP variant el nombre de threads. És més lògic, ja que alhora de valorar els speed-ups, en aquesta secció, el que relament vull saber és si val la pena muntar un clúster per treure més rendiment que amb un sol node. Les mesures amb 1 sol procès MPI, (que no estan reportades a la memòria), són semblants a les del codi serial amb OpenMP però, amb menys rendiment, causat per l'overhead d'usar una interfície addicional que és MPI.

- Mida de l'entrada en bytes: 32 MB, Nombre d'elements: 2048.

#OpenMP/#MPI	0	2	4
1	64,11	45,53	31,14
2	68,19	28,44	31,20
4	68,32	29,94	31,58
10	68,68	32,52	33,40

Taula 7.1: Mesures de temps de la factorització LU amb mida d'entrada de 32 MB

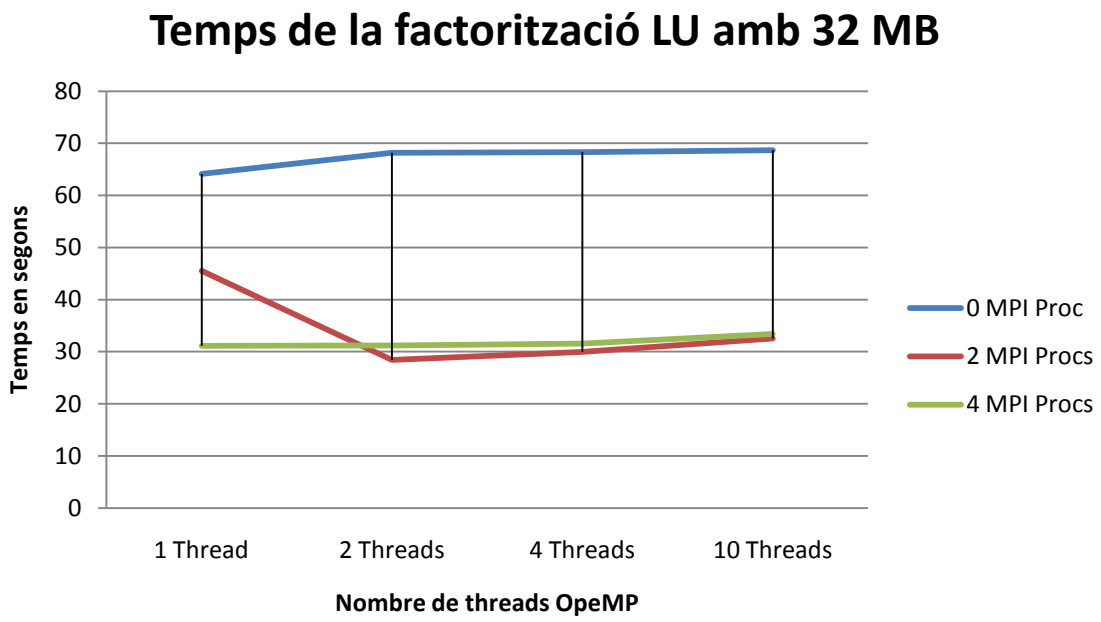

Figura 7.8: Gràfica dels temps de la factorització LU amb mida d'entrada de 32 MB

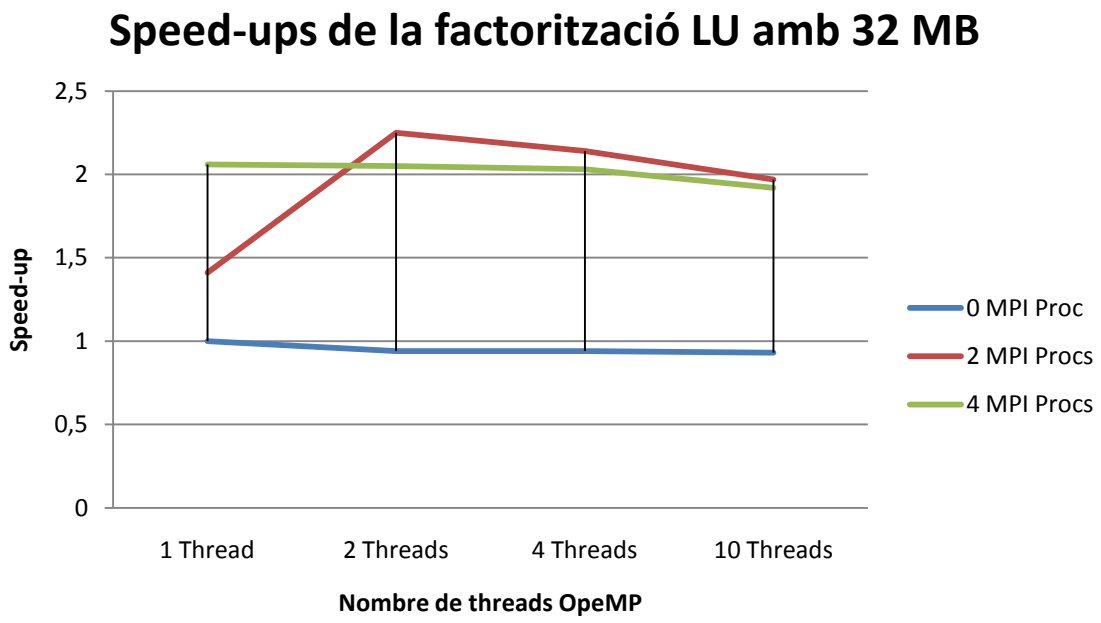

Figura 7.9: Gràfica dels speed-ups de la factorització LU amb mida d'entrada de 32 MB

- Mida de l'entrada en bytes: 4 MB, Nombre d'elements: 724.

#OpenMP/#MPI	0	2	4
1	2,94	2,42	3,61
2	2,62	2,06	2,65
4	2,72	2,09	2,53
10	2,97	2,39	2,86

Taula 7.2: Mesures de temps de la factorització LU amb mida d'entrada de 4 MB

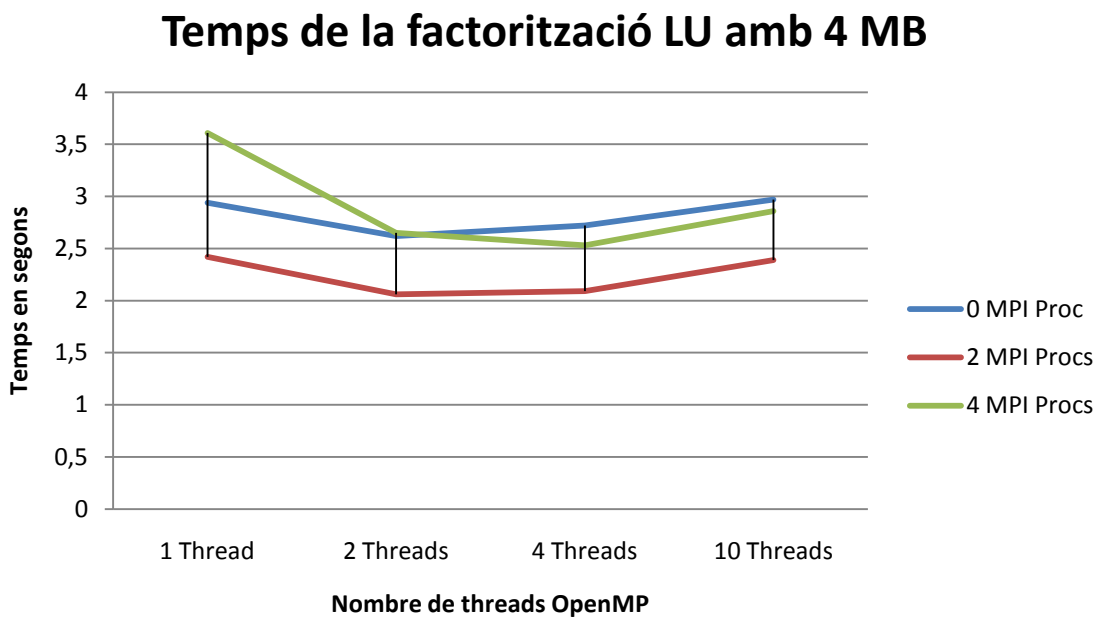

Figura 7.10: Gràfica dels temps de la factorització LU amb mida d'entrada de 4 MB

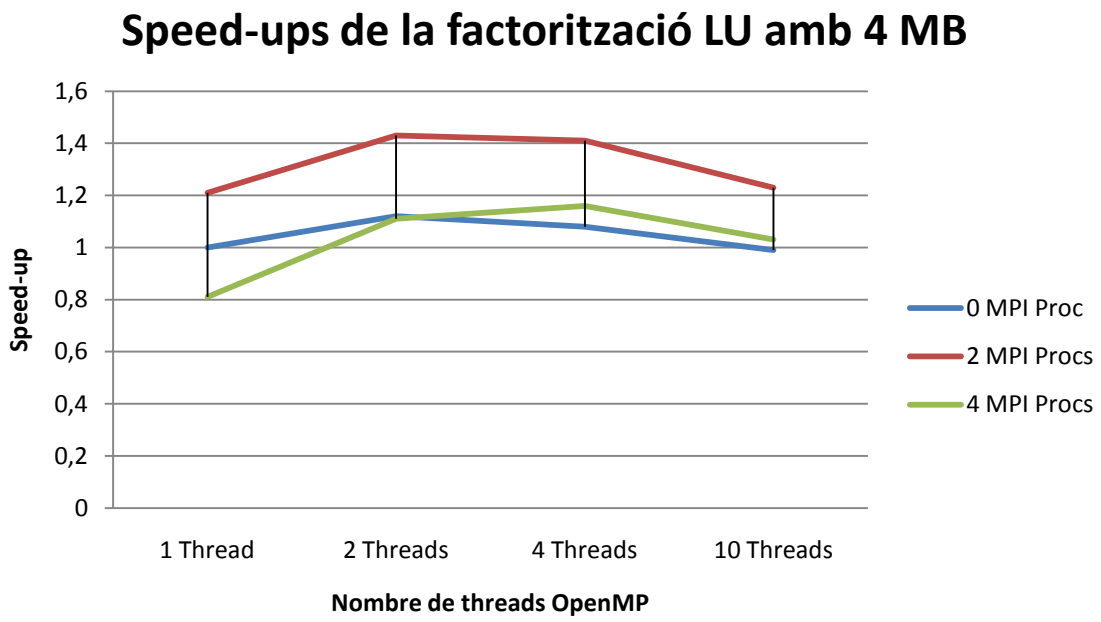

Figura 7.11: Gràfica dels speed-ups de la factorització LU amb mida d'entrada de 4 MB

- Mida de l'entrada en bytes: 2 MB, Nombre d'elements: 512.

#OpenMP/#MPI	0	2	4
1	1,02	1,51	1,72
2	1,07	1,39	1,65
4	1,07	1,45	1,63
10	1,22	1,51	1,70

Taula 7.3: Mesures de temps de la factorització LU amb mida d'entrada de 2 MB

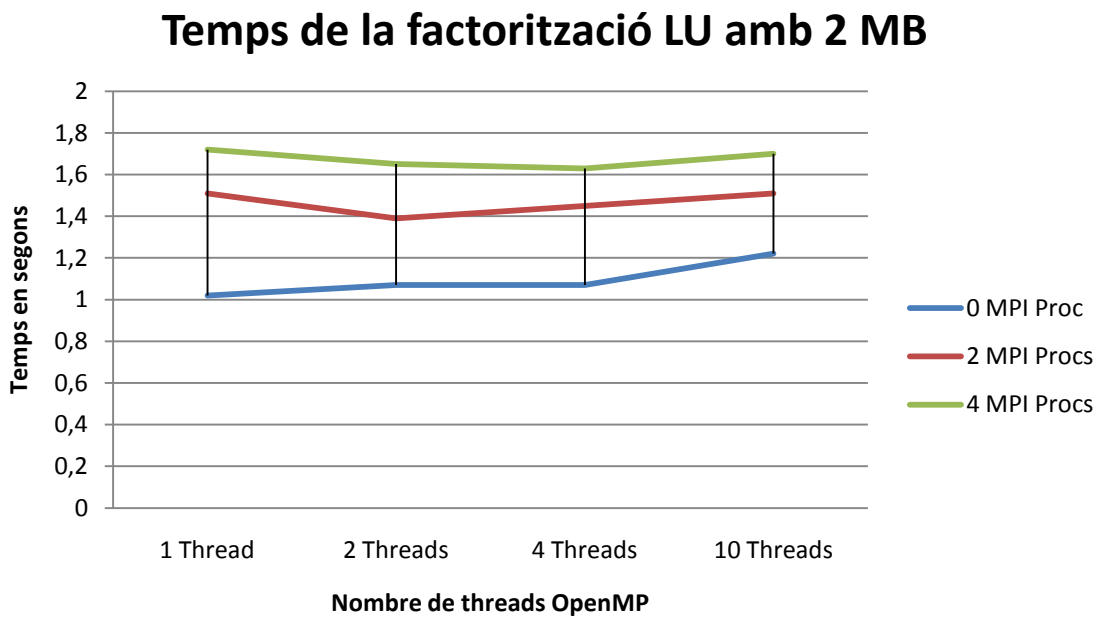

Figura 7.12: Gràfica dels temps de la factorització LU amb mida d'entrada de 2 MB

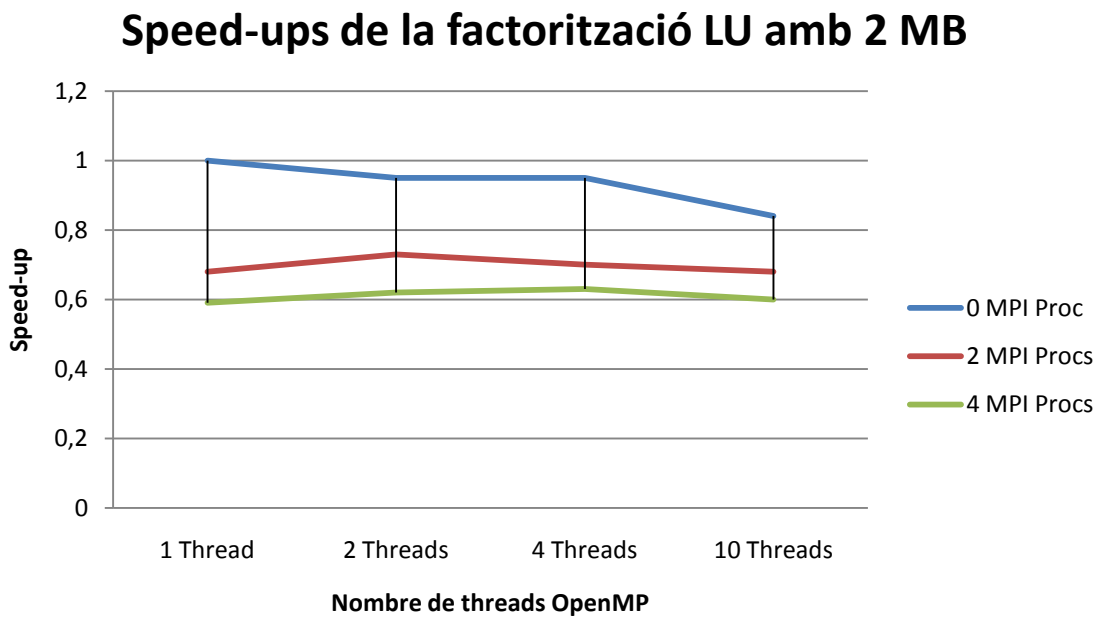

Figura 7.13: Gràfica dels speed-ups de la factorització LU amb mida d'entrada de 2 MB

- Mida de l'entrada en bytes: 256 KB, Nombre d'elements: 181.

#OpenMP/#MPI	0	2	4
1	0,08	1,51	1,80
2	0,11	1,50	1,50
4	0,12	1,46	1,64
10	0,17	1,49	1,99

Taula 7.4: Mesures de temps de la factorització LU amb mida d'entrada de 256 KB

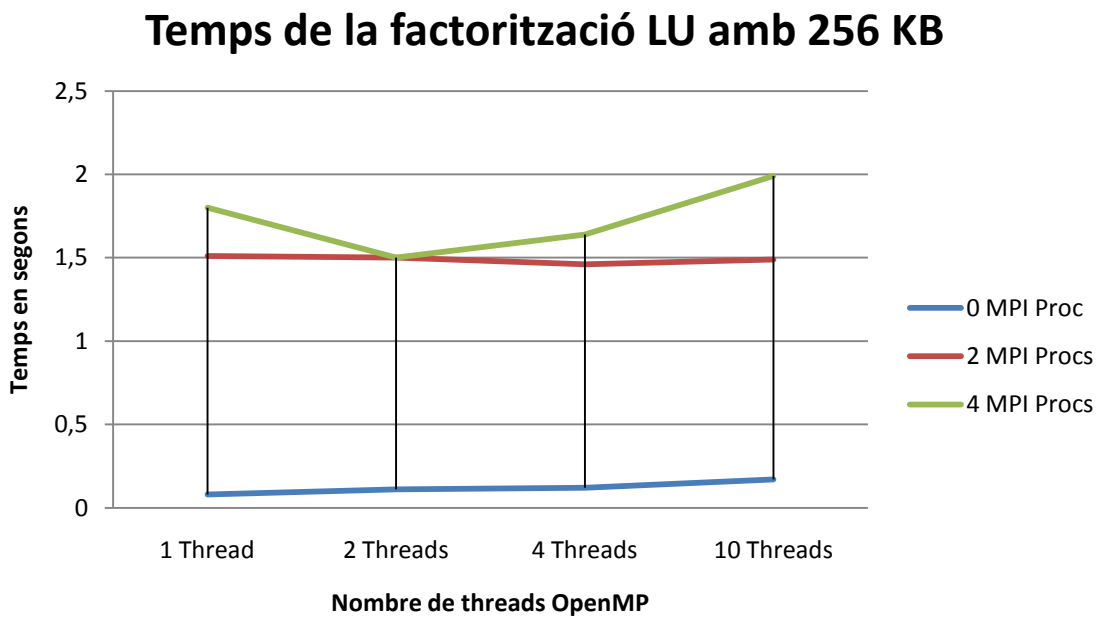

Figura 7.14: Gràfica dels temps de la factorització LU amb mida d'entrada de 256 KB

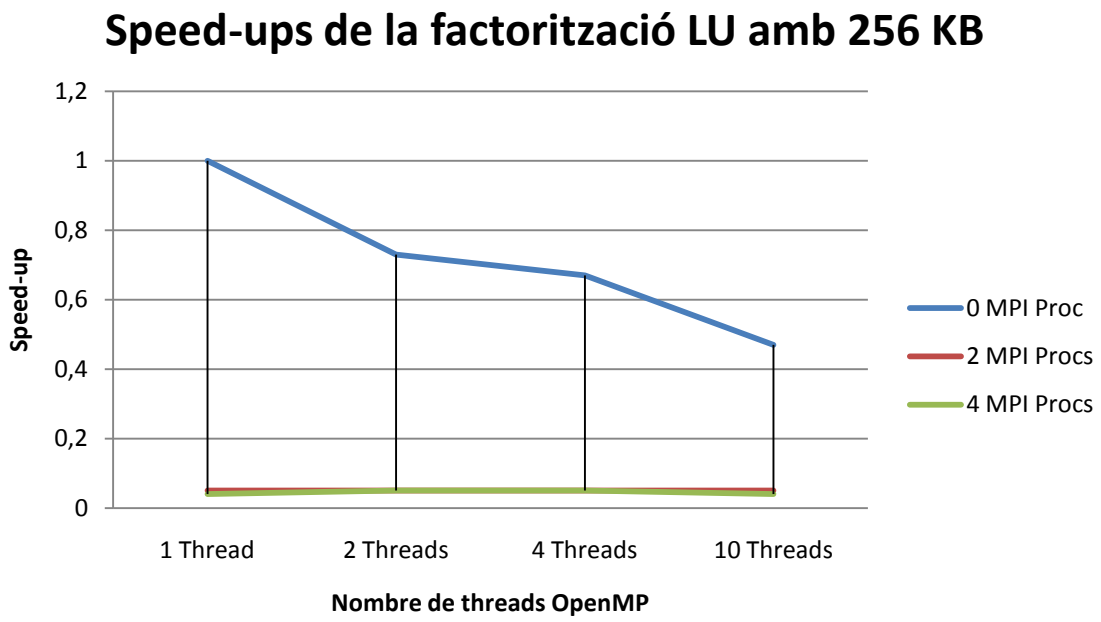

Figura 7.15: Gràfica dels speed-ups de la factorització LU amb mida d'entrada de 256 KB

- Millor speed-up per a cada mida d'entrada.

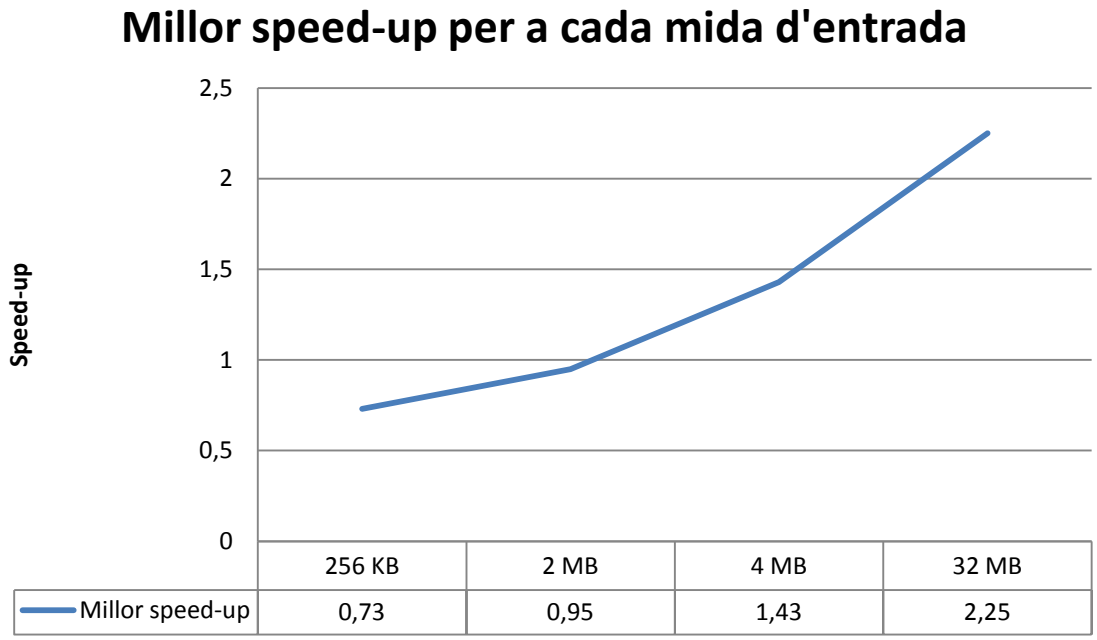

Figura 7.16: Millor speed-up en funció de la mida de l'entrada per la factorització LU

7.2.4 Valoració dels resultats

Els resultats de les execucions ens mostren que per mides d'entrada peites com ara 256 KB i 2 MB l'ús de MPI relentitza l'execució i que no és fins a mides a partir de 4 MB on hi ha speed-up degut a l'ús de MPI usant els dos nodes.

Veiem una diferència molt grossa entre els temps de 4 MB i 32 MB que es correspon amb que el cost temporal de l'algorisme és de O(n^3).

La mida del bloc usada ha estat de 150 elements, és a dir de 176 KB; més que el primer nivell de memòria cau, (32 KB), però inferior al segon nivell, (256 KB). Pel que no hauriem de tenir gaires problemes de localitat temporal.

Crida molt l'atenció que els temps millors per a l'execució de 4 MB i de 32 MB siguin amb 2 processos MPI i 2 threads. On cada node, (placa), té un process MPI i cada processador un thread a cada nucli. És a dir que no està escalant bé el gra fi, tot i tenir unes mides de blocs adients, el que fa pensar que en aquest cas o bé no tenim speed-up per usar més threads o bé aquest no ens està fent guanyar prou temps com per permetrens l'overhead de la gestió dels threads per OpenMP. Una altre qüestió seria provar altres planificacions d'OpenMP.

7.3 Factorització Cholesky

7.3.1 Algorisme

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 10.

7.3.2 Dependències de dades

7.3.2.1 Algorisme seqüencial

```
for (k=0; k<N ; k++){
            A[k][k]= sqrt(A[k][k]);
        for (j=k+1; j<N; j++){
                    A[k][j] = A[k][j] / A[k][k];
        }
        for(i=k+1; i<N; i++){
                    for(j=i; j<N; j++){
                A[i][j]= A[i][j] - A[k][i] * A[k][j];
                    }
        }
}
```

Codi 7.2: Algorisme de factorització Cholesky sequencial

Iteració 1: k=0

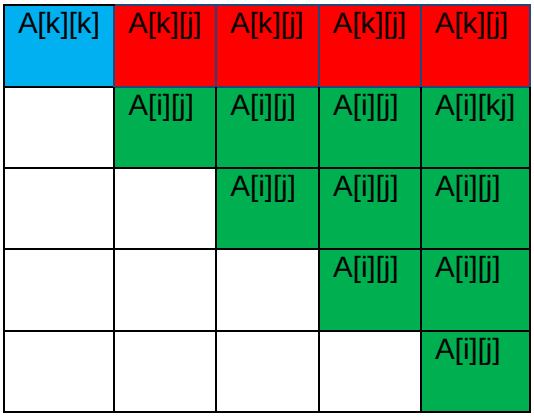

Figura 7.17: Iteració 1 de l'algorisme Cholesky sequencial

Iteració 2: k=1

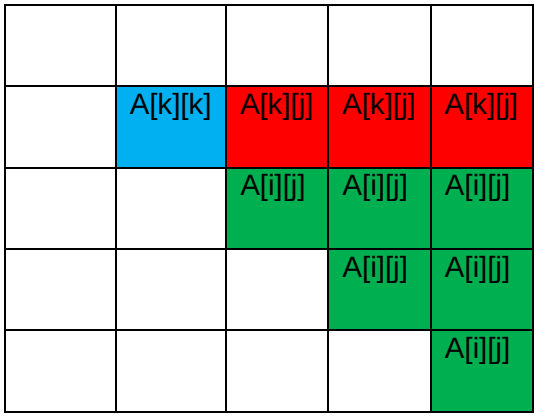

Figura 7.18: Iteració 2 de l'algorisme Cholesky sequencial

En l'algorisme de la factorització Cholesky el primer que hem de calcular és l'arrel quadrada de l'element de la cantonada superior dreta de la matriu. Posteriorment la divisió dels elements de la primera fila respecte l'element de la cantonada superior dreta.

Finalment es calcula el mòdul dels elements de la matriu diagonal superior a partir de la fila iteració +1 respecte els elements variables A [k][i] i A [k][j].

7.3.2.2 Algorisme per blocs

En l'algorisme per blocs per MPI seguim la mateixa metodologia que en la factorització LU, veure apartat 7.2.2.2 de la memòria.

Aquí també faig ús de la tècnica de fer la mida del bloc variable per tal d'evitar la inanició dels processadors. Veiem com en la factorització Cholesky no tractem la matriu diagonal inferior.

Iteració 1:

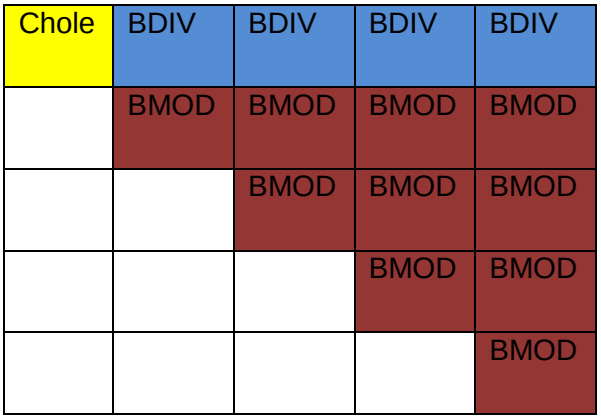

Figura 7.19: Iteració 1 de l'algorisme Cholesky amb blocking

Iteració 2:

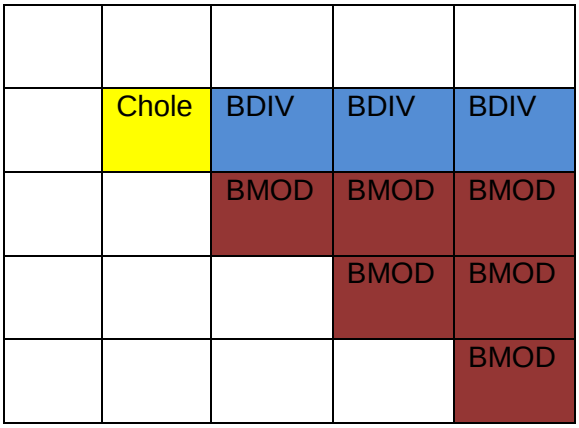

Figura 7.20: Iteració 2 de l'algorisme Cholesky amb blocking

En la factorització Cholesky primer hem de calcular el bloc Chole, el qual no té dependències amb cap altre bloc i que per tant no hem d'enviar. Posteriorment el mateix processador que ha calculat el bloc Chole, calcula els blocs BDIV de manera tan bon punt es té calculat un bloc s'envia perquè els altres processadors puguin calcular el seu bloc BMOD corresponent a la columna del bloc BDIV enviat.
En el tractament dels blocs situats en la diagonal de la matriu, per seguir l'algorisme seqüencial estrictament hauríem de tractar-los de manera diferent a la resta recorrent-los només en la seva meitat diagonal superior. No obstant, com que el cost de comprovar si el bloc és o no de la diagonal de la matriu suposaria inserir comprovacions en el gra gros i per consegüent un retard, he decidit tractar-los com els altres i ignorar el resultat referent als elements que es calculen i que no formen part del resultat.

Per mantenir les operacions que fa l'algorisme Cholesky seqüencial en el bloc Chole hem de fer l'algorisme de Cholesky seqüencial, en els blocs BDIV hem de fer el l'algorisme de Cholesky seqüencial sense fer l'arrel de l'element de la cantonada superior dreta i en els blocs BMOD també l'algorisme seqüencial però, només fent els mòduls respecte els elements que es troben en el bloc BDIV corresponent .

7.3.3 Mesures de temps i gràfiques

Per a les mesures de temps amb MPI he variat el nombre de processos MPI i el nombre de threads OpenMP per a determinades mides de dades d'entrada. Algunes combinacions de nombre de processos MPI i de nombre de threads OpenMP que en principi pot semblar que no tenen sentit, com veurem, a la pràctica fan els millors temps.

MPICH2 ha mapejat els processos MPI de manera que els processos imparells s'allotjen en el processador de la placa màster i els processos parells s'allotjen en la placa esclava.

En les següents taules la primera columna indica els threads OpenMP mentre que la primera fila indica els processos MPI.

- Mida de l'entrada en bytes: 32 MB, Nombre d'elements: 2048.

#MPI/#OpenMP	0	2	4
1	33,07	20,15	22,03
2	33,57	15,84	22,69
4	33,74	17,75	24,00
10	34,40	21,41	24,70

Taula 7.5: Mesures de temps de la factorització Cholesky amb mida d'entrada de 32 MB

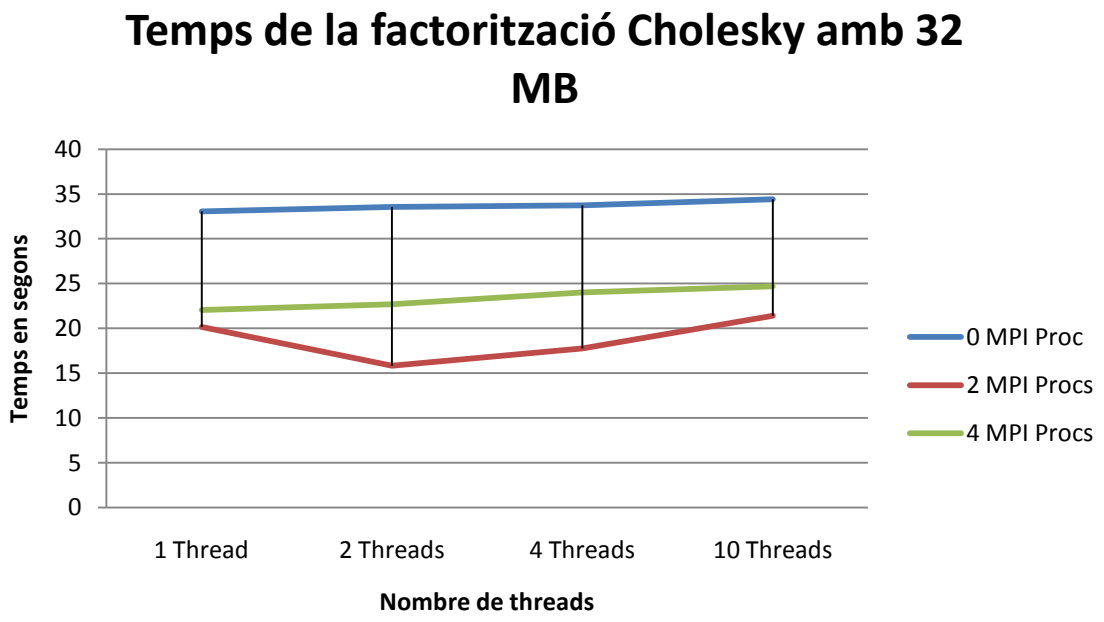

Figura 7.21: Gràfica dels temps de la factorització Cholesky amb mida d'entrada de 32 MB

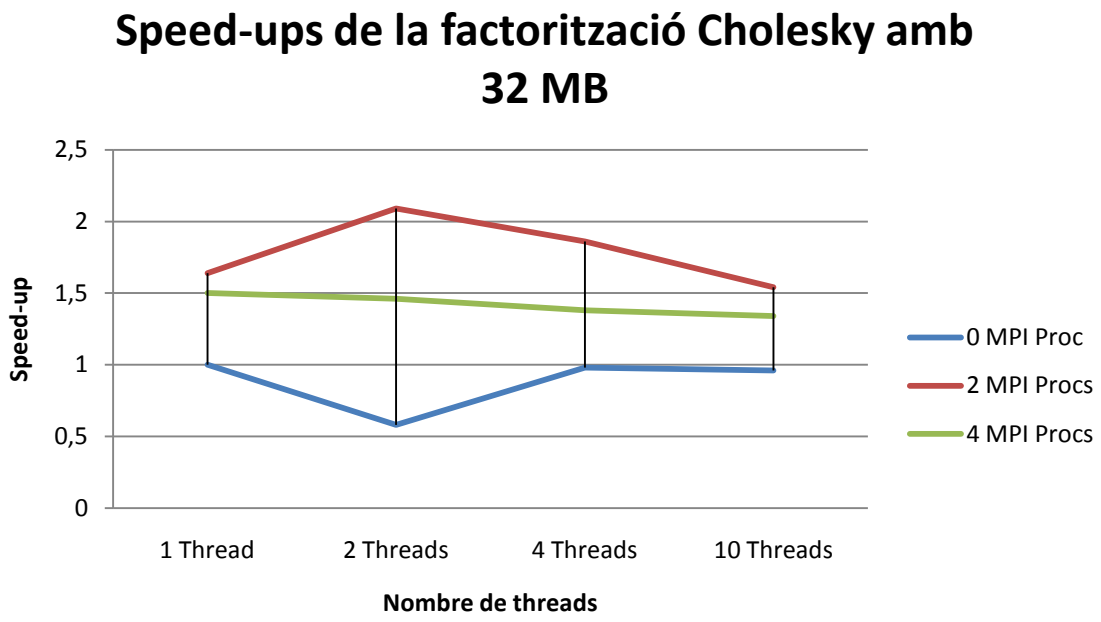

Figura 7.22: Gràfica dels speed-ups de la factorització Cholesky amb mida d'entrada de 32 MB

- Mida de l'entrada en bytes: 4 MB, Nombre d'elements: 724.

#MPI/#OpenMP	0	2	4
1	1,49	1,45	2,68
2	1,48	1,51	2,38
4	1,55	1,56	2,28
10	1,76	1,81	2,70

Taula 7.6: Mesures de temps de la factorització Cholesky amb mida d'entrada de 32 MB

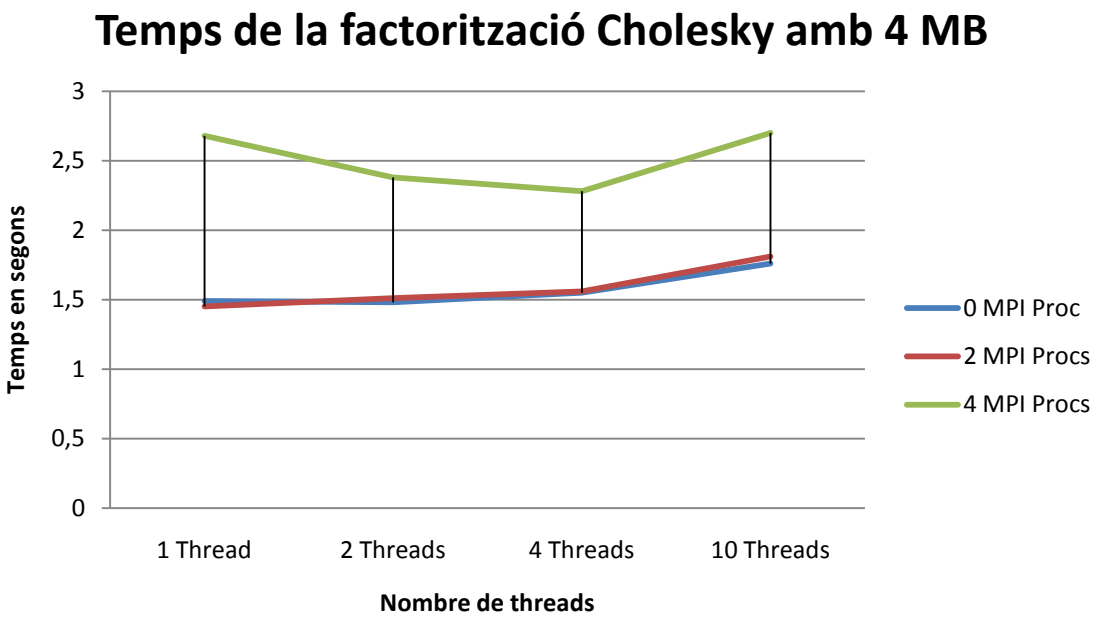

Figura 7.23: Gràfica dels temps de la factorització Cholesky amb mida d'entrada de 4 MB

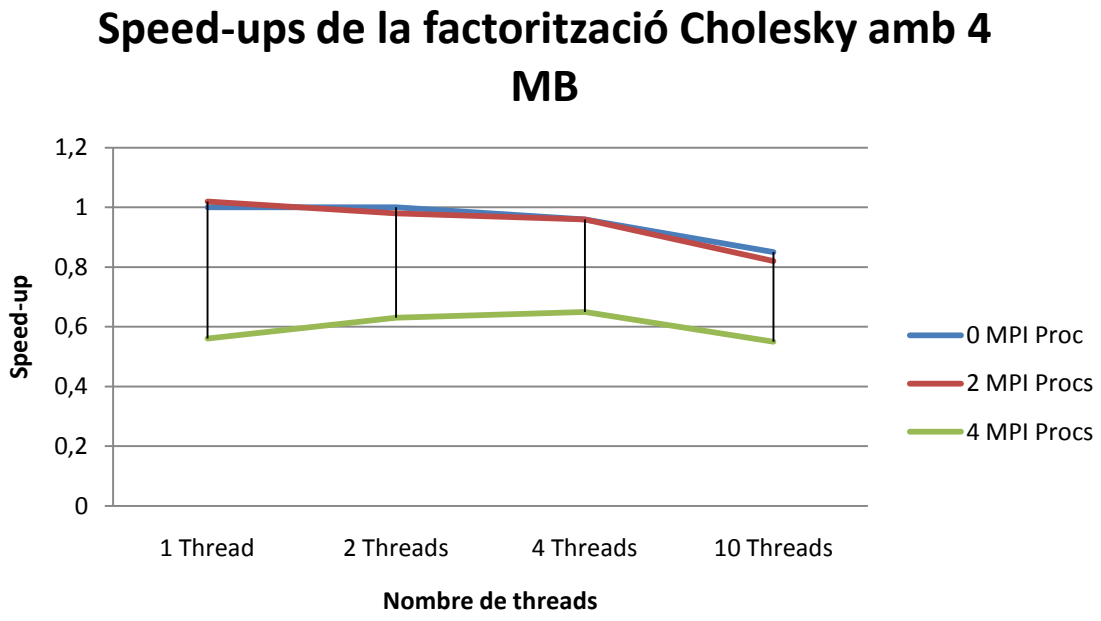

Figura 7.24: Gràfica dels speed-ups de la factorització Cholesky amb mida d'entrada de 4 MB

- Mida de l'entrada en bytes: 2 MB, Nombre d'elements: 512.

#MPI/#OpenMP	0	2	4
1	0,54	0,72	1,58
2	0,53	0,77	1,31
4	0,57	0,80	1,26
10	0,72	0,95	1,41

Taula 7.7: Mesures de temps de la factorització Cholesky amb mida d'entrada de 2 MB

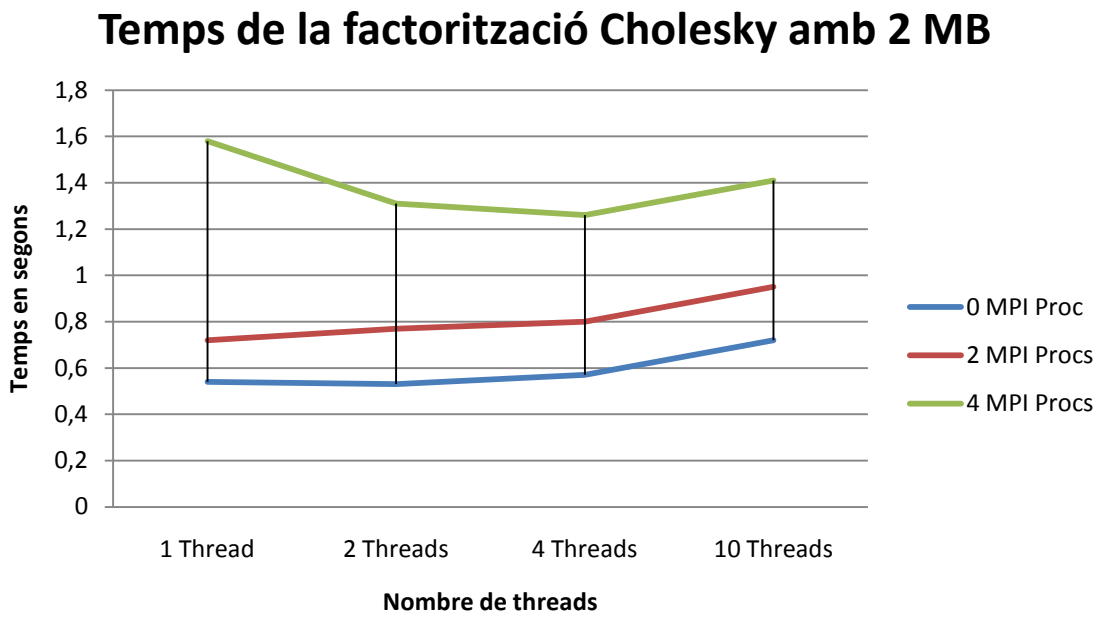

Figura 7.25: Gràfica dels temps de la factorització Cholesky amb mida d'entrada de 2 MB

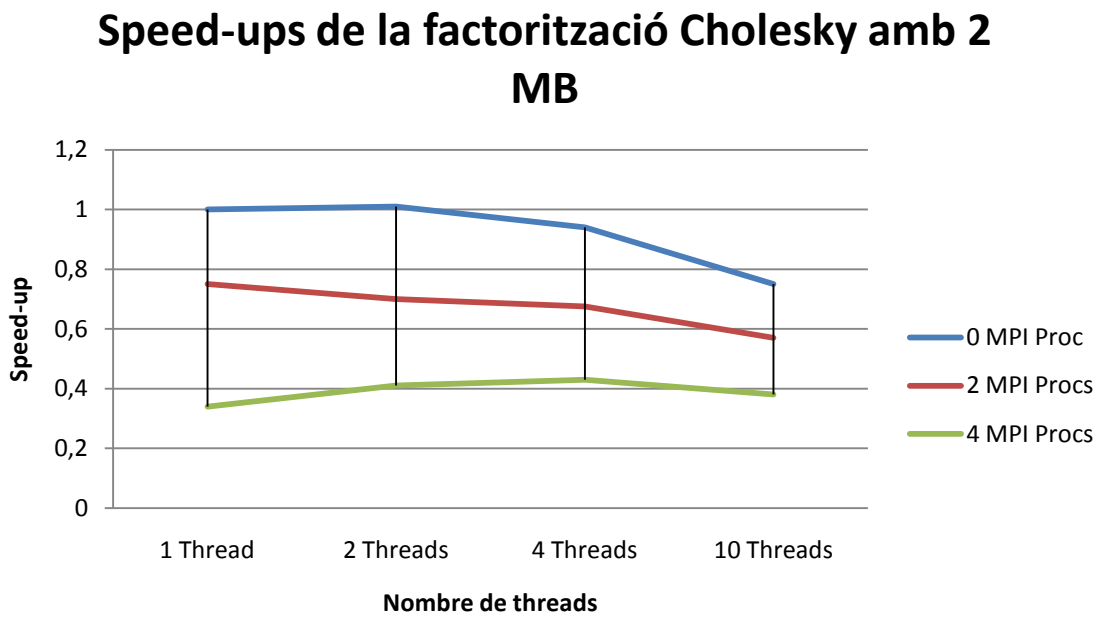

Figura 7.26: Gràfica dels speed-ups de la factorització Cholesky amb mida d'entrada de 2 MB

- Mida de l'entrada en bytes: 256 KB, Nombre d'elements: 181.

#MPI/#OpenMP	0	2	4
1	0,06	0,72	1,46
2	0,08	0,75	1,28
4	0,09	0,80	1,32
10	0,16	0,95	1,37

Taula 7.8: Mesures de temps de la factorització Cholesky amb mida d'entrada de 32 MB

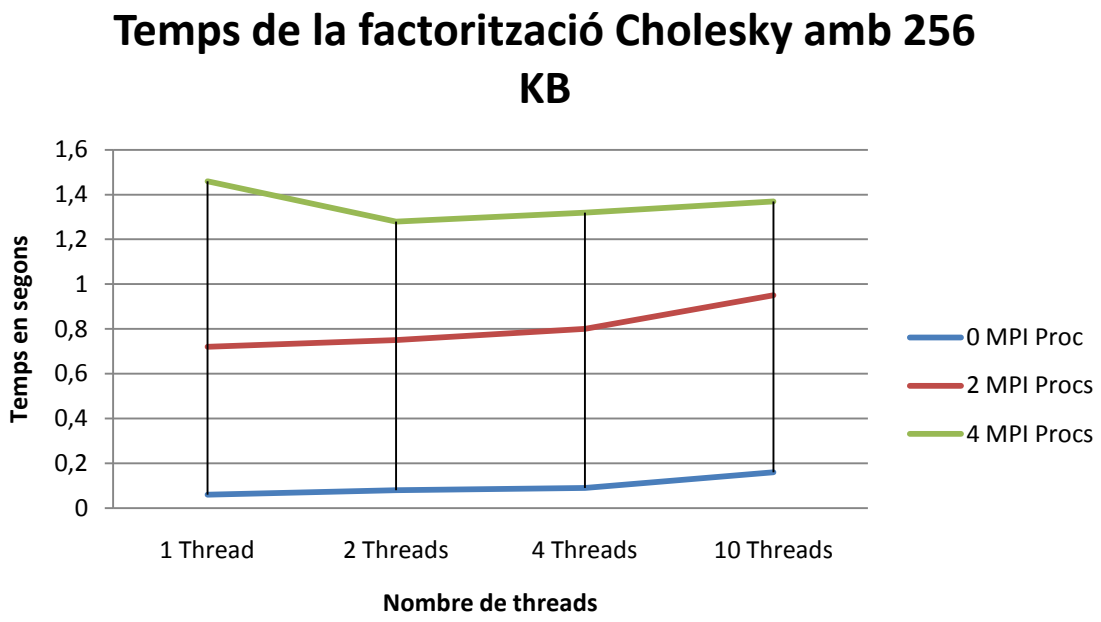

Figura 7.27: Gràfica dels temps de la factorització Cholesky amb mida d'entrada de 256 KB

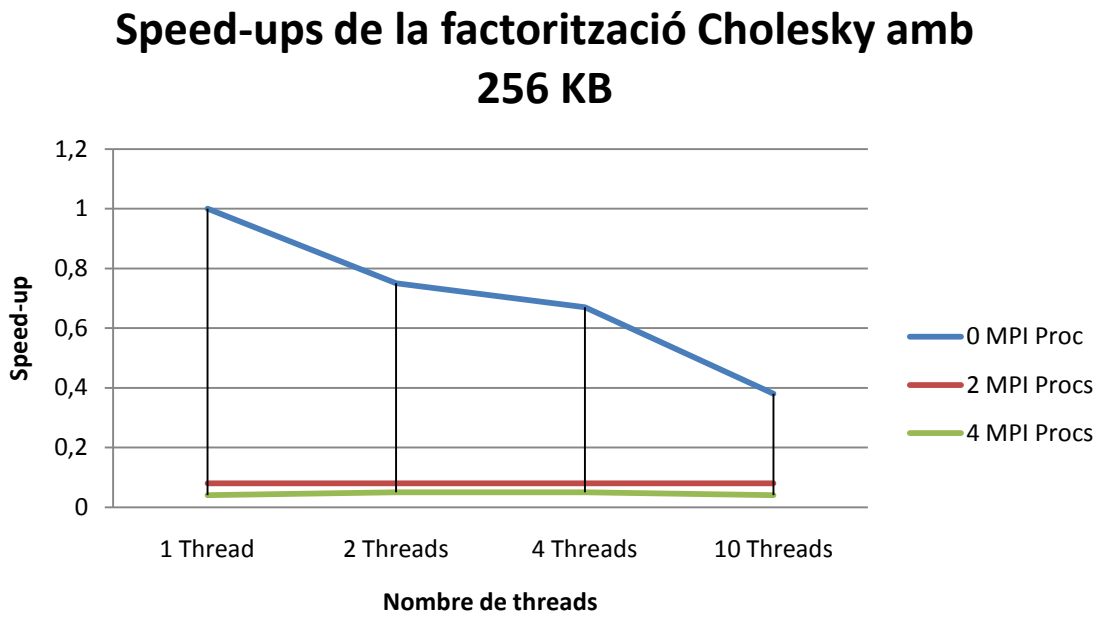

Figura 7.28: Gràfica dels speed-ups de la factorització Cholesky amb mida d'entrada de 256 KB

- Speed-up òptim en funció de la mida de l'entrada.

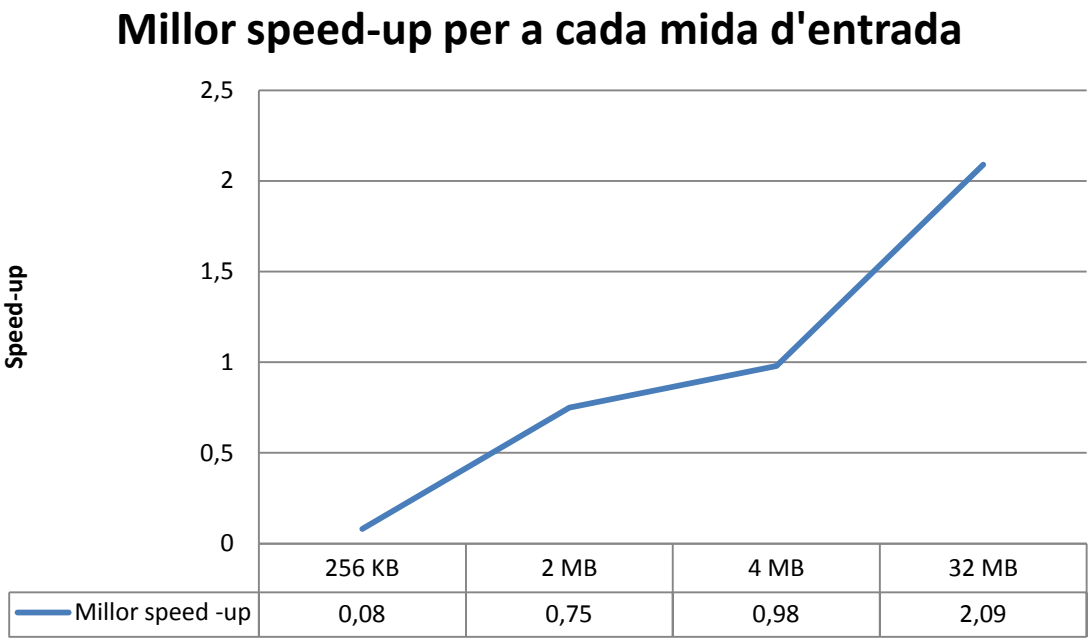

Figura 7.29: Millor speed-up en funció de la mida de l'entrada per la factorització Cholesky

7.3.4 Valoració dels resultats

El rendiment del sistema en l'algorisme de Cholesky en aquest experiment és casi igual al comporament al comportament en l'algorisme LU. No obstant, per a la mida d'entrada de 4MB el rendiment òptim és usant 2 processos MPI, un en cada node, i usant un sol thread OpenMP. És un cas curiós on és més rápida la comunicació entre nodes que la gestió dels threads dins del node i de la seva coherencia en la jerarquía de memòria.

En general els temps són inferiors perquè només s'opera amb la meitat diagonal superior de la matriu no obstant, el cost temporal de l'algorisme segueix sent O(n^3).

El millor speed-up en el Cholesky amb 2 Processos MPI I dos threads és de 2,09 mentre que en el LU és de 2,25. Això probablement és degut a que en la paral·lelització de l'algorisme Cholesky en aquesta implementació, les primeres columnes de blocs recorregudes de la matriu, tenen poques files de blocs a repartir entre els processadors i per tant hi ha menys paral·lelització .

7.4 Mesures de la xarxa

Una de les mesures bàsiques en un clúster és comprovar el rendiment de la xarxa que el connecta. De fet és el que diferencia un clúster de segona categoria d'un supercomputador.

7.4.1 Ping-Pong

Una prova molt usada és fer córrer un algorisme de ping-pong. El que fa és enviar un paquet de dades des d'un node emissor a un receptor, el receptor torna el paquet a l'emissor i aquest últim torna a enviar un altre paquet més gros. A partir del que triga a tornar cada paquet es treuen mesures de l'ample de banda i de la latència de la xarxa, [21], [38*].

7.4.1.1 Codi del programa

L'algorisme es troba en l'annex, secció "codis de programa usats", codi 11.

7.4.1.2 Resultats

Les següents taules mostren els resultats per a l'execució del mateix algorisme de ping-pong amb crides MPI bloquejants, crides asíncrones i comunicació bidireccional.

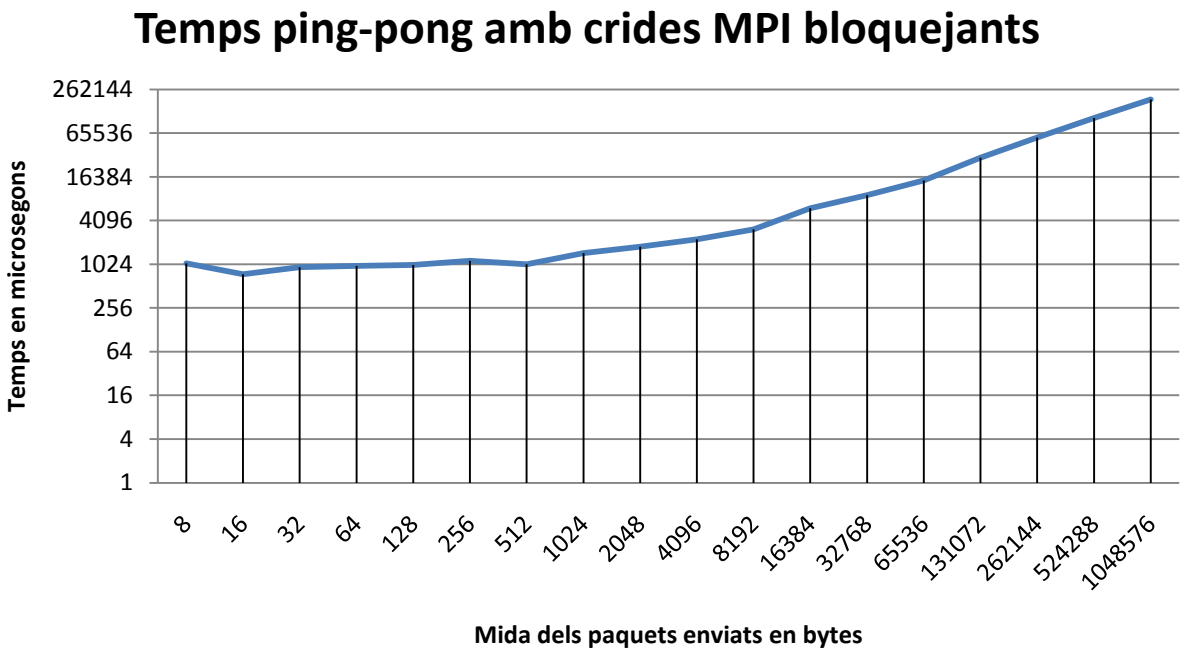

Figura 7.30: Resultats de l'execució de l'algorime de ping-pong amb crides MPI bloquejants

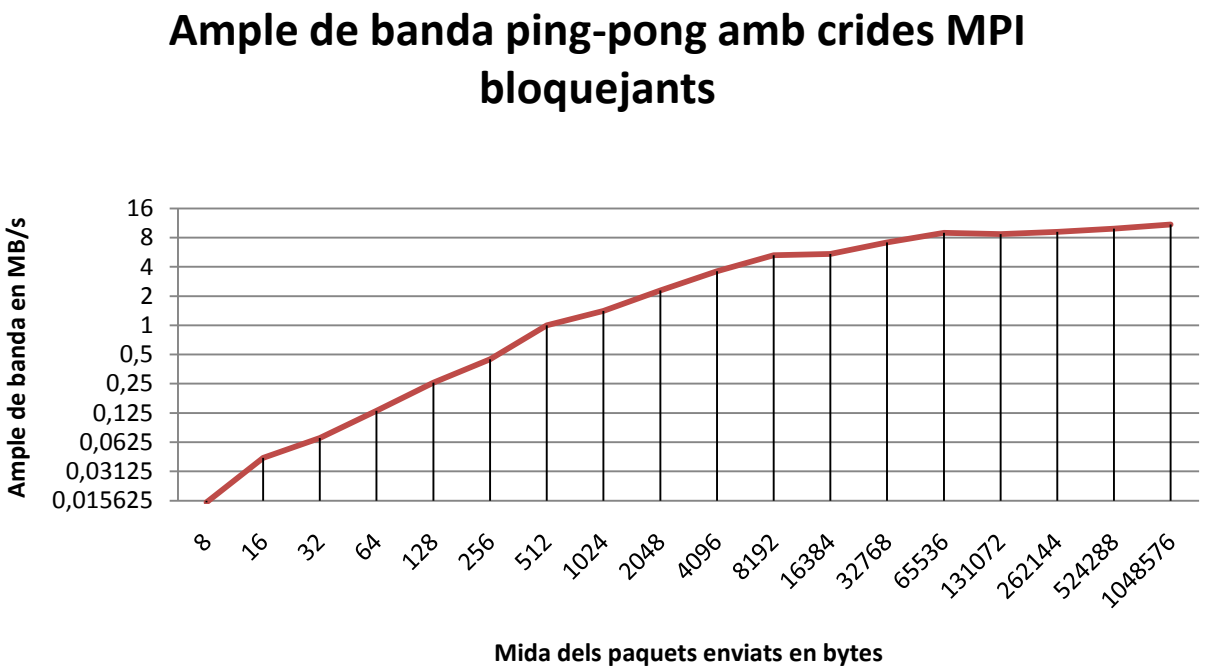

Figura 7.31: Resultats de l'execució de l'algorime de ping-pong amb crides MPI bloquejants

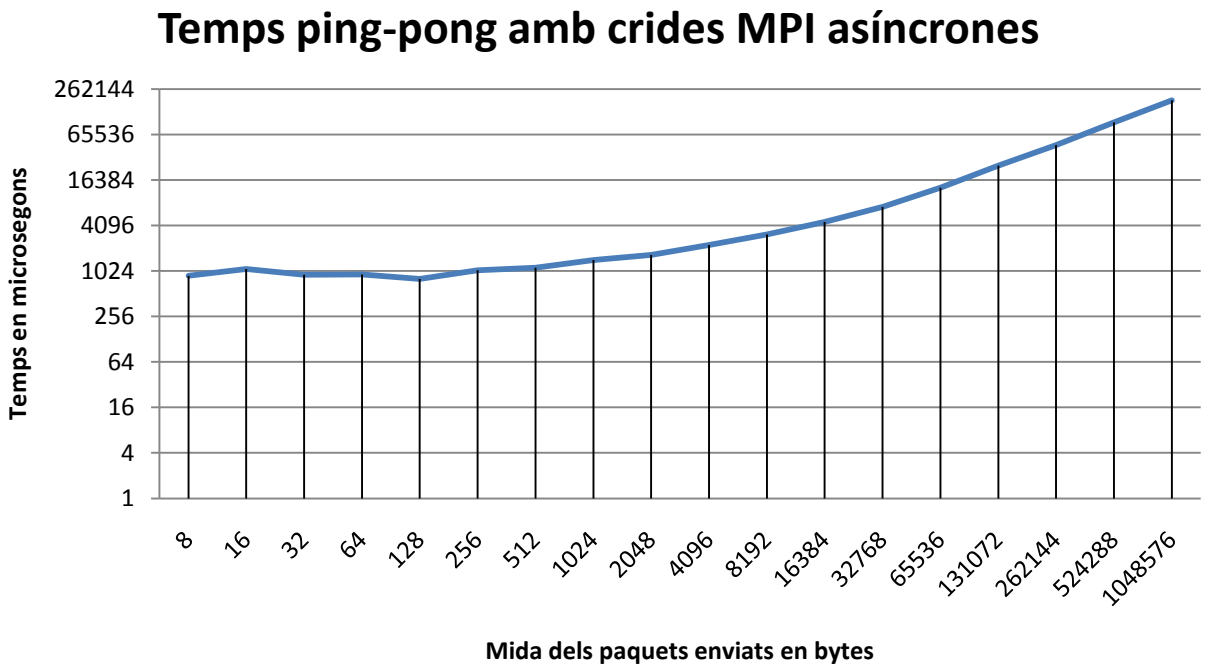

Figura 7.32: Resultats de l'execució de l'algorisme de ping-pong amb crides MPI asíncrones

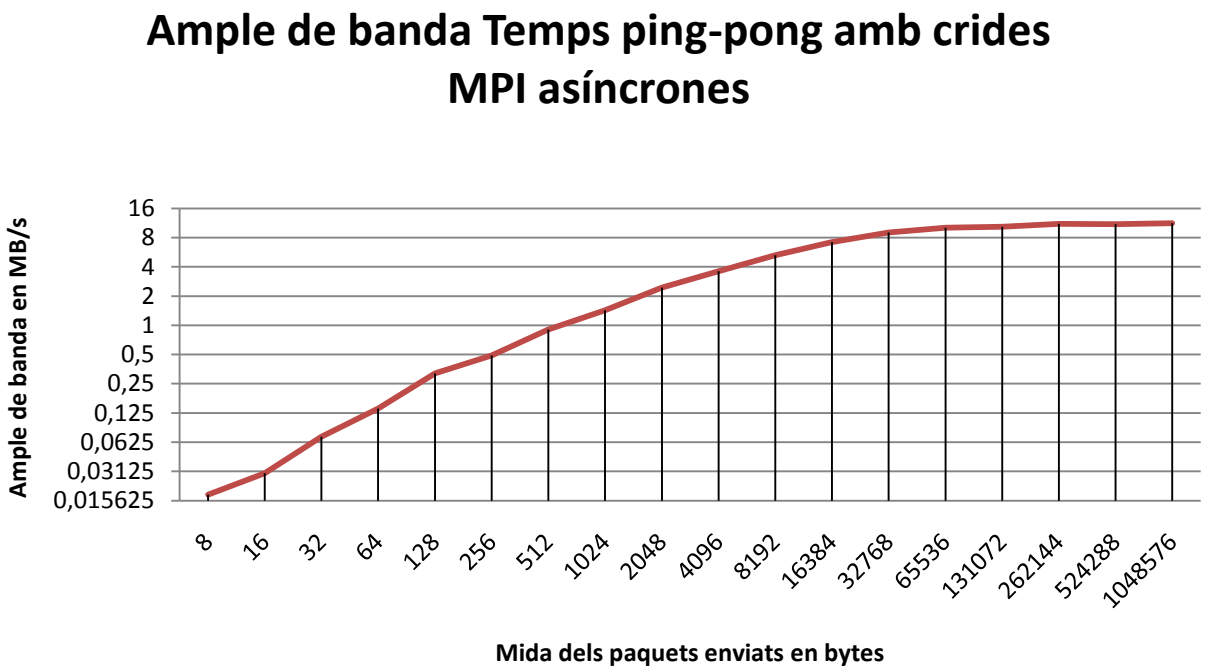

Figura 7.33: Resultats de l'execució de l'algorisme de ping-pong amb crides MPI asíncrones

7.4.1.3 Valoracions del resultats

La velocitat màxima assolida ha estat 11,31 MB/segon; si tenim en compte que he usat Ethernet de 100 Mbps, que és el mateix que 12,5 MB/segon, resulta que el màxim rendiment assolit ha estat del 90,48%.

Pel que fa la latència mínima, si la prenem com la meitat del round trip delay time mínim, aleshores hem de fer 744 microsegons / 2 = 372 microsegons, molt lluny dels 6-3 microsegons que es poden aconseguir amb fibra òptica.

Si mirem la progressió de les latències i dels amples de banda són normals, a més mida de dades més ample de banda i més latència.

En quant a les diferències entre el ping-pong bloquejant, asíncron i asíncron bidireccional, trobem que no hi ha gaire millora del asíncron respecte del bloquejant ja que el còmput que es fa entremig és una simple suma.

7.4.2 NetPIPE

NetPipe és una eina per mesurar el rendiment d'una xarxa sota una varietat de condicions independentment del protocol. Fa tests ping-pong que llancen missatges d'un procés a un altre que reboten i arriben de nou a l'emissor. Les latències són calculades dividint entre dos el RTT, (round trip delay time), per a missatges més petits de 64 bytes, [55*], [56*], [57*].

7.4.2.1 Instal·lació

- Descarregar NetPIPE i descomprimir-lo preferiblement en algun lloc on tinguem permisos d'escriptura.
  - http://www.scl.ameslab.gov/Projects/NetPIPE/
- Instal·lar l'interpret csh.
  - sudo apt-get install csh
- Instal·lar gnuplot per fer les gràfiques.
  - sudo apt-get install gnupot
- Editar el makefile.
  - MPI2_LIB=/usr/lib/libmpich.a
  - MPI_INC=/usr/include
- Si volem probar les comunicacións a dues bandes, (crides MPI_put() i MPI_get()), que ofereix la implementació de MPI 2 compilem amb:
  - make mpi-2
- Si volem probar fer les probes amb les crides MPI_send() i MPI_recv() típiques de MPI compilem amb:
  - make mpi

7.4.2.2 Ús

- Aixecar el clúster.
  - mpdboot –n 2 –f /home/clusty/mpd.hosts
- Córrer el programa.
  - mpiexec –n 2 ./NPmpi
- Fer les gràfiques.
  - gnuplot
  - set logscale x (ja que la mida del bloc usat s'incrementa en exponencialment).
  - set xlabel "unitats eix x"
  - set ylabel "unitats eix y"
  - set title "titol del gràfic"
  - plot "/pathfinsalfitxerdesortida" using 1:2
  - set terminal postscript color solid
  - set output "home/clusty/Escritorio/myplot.ps"
  - replot

7.4.2.3 Resultats

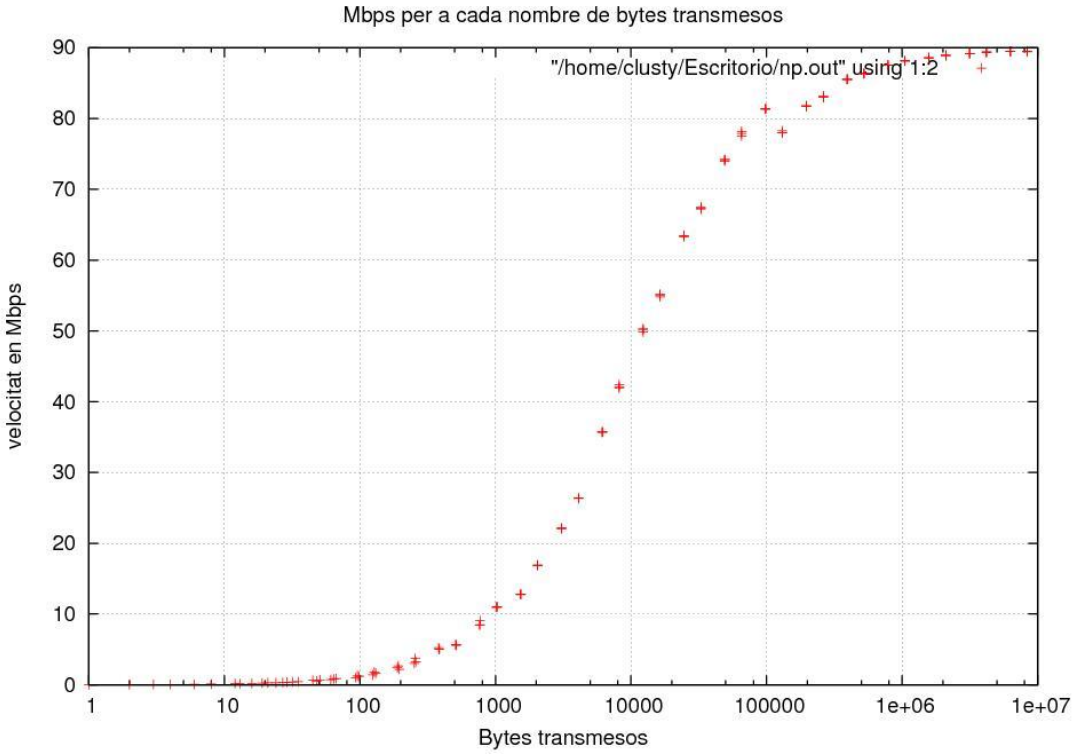

Figura 7.34: Mbps per a cada nombre de bytes transmesos amb crides MPI send recv

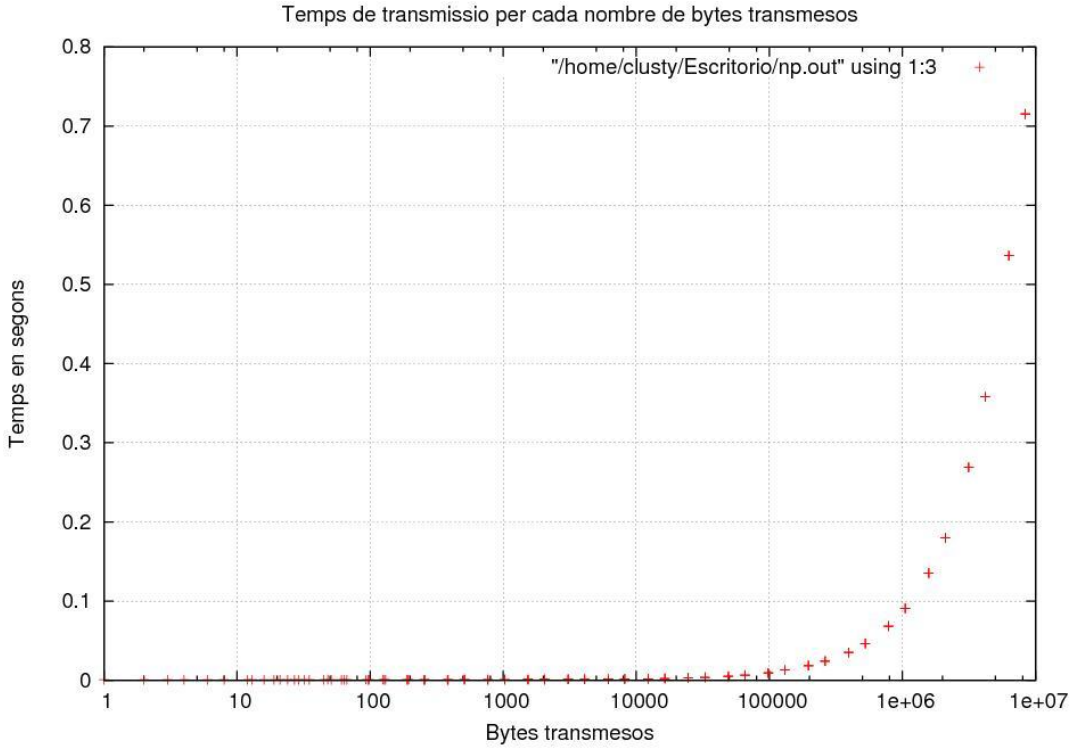

Figura 7.35: Temps per a cada nombre de bytes transmesos amb crides MPI send recv

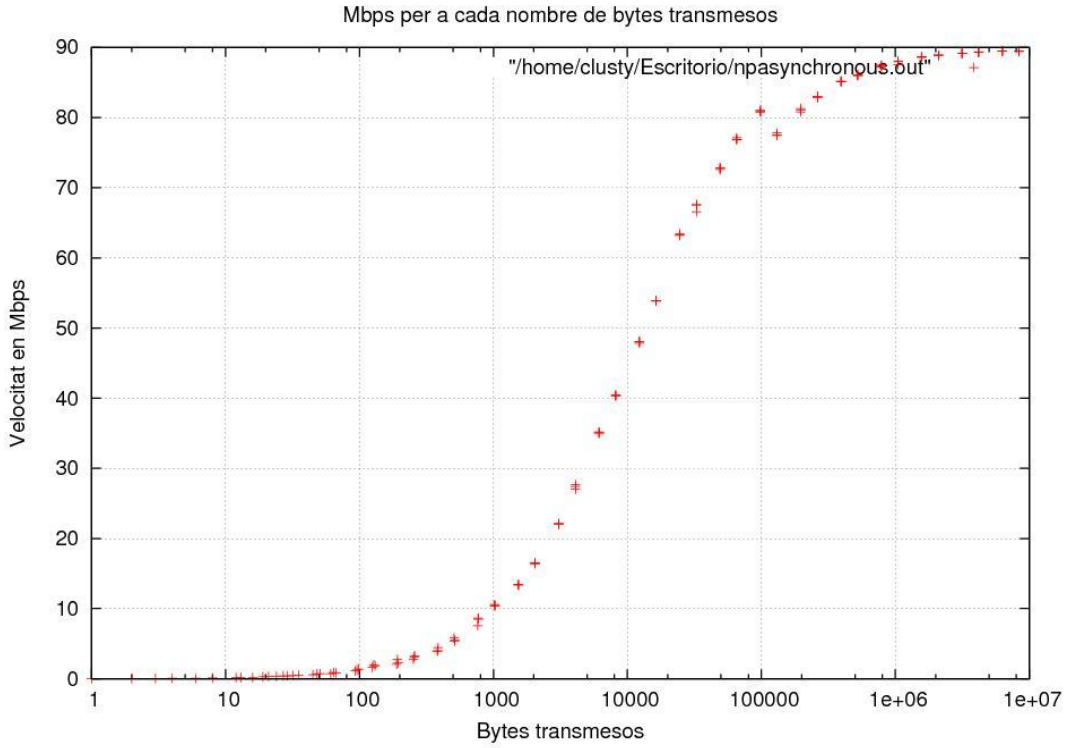

Figura 7.36: Mbps per a cada nombre de bytes transmesos amb crida MPI recv asíncrona

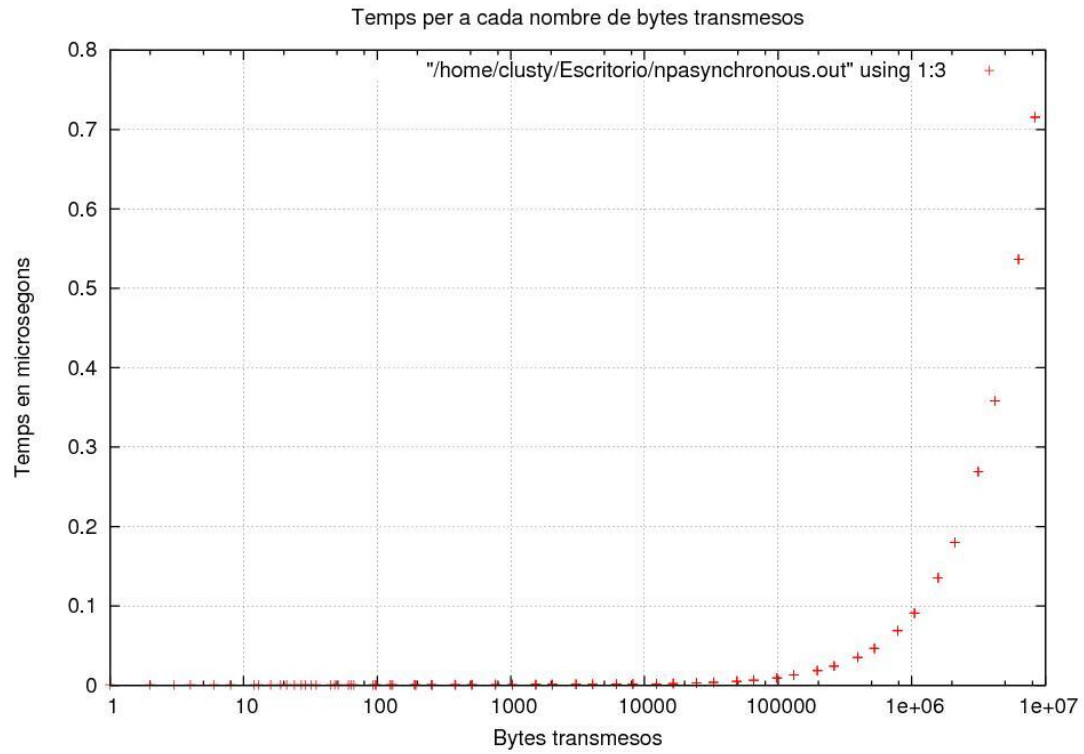

Figura 7.37: Temps per a cada nombre de bytes transmesos amb crida MPI recv asíncrona

7.4.2.4 Valoració dels resultats

Podem observar a simple vista com en l'execució del ping-pong, en els dos casos, recepció bloquejant i recepció asíncrona, els resultats són molt semblants.

Veiem com la corba de l'ample de banda segons la mida de les dades enviades en cada missatge segueix una relació exponencial de 1 a 5500 bytes,(5,5 KB), que passa a ser lineal dels 5500 bytes,(5,5 KB) als 55000 bytes, (53,7 KB), per acabar sent logarítmica dels 55000 bytes, (5,5 KB), als 10^7 bytes, (9,54 MB).

En quan al temps consumit s'aprecia una corba exponencial que comença a elevar-se a partir dels 10000 bytes, (9,76 KB).

Aquest comportament del rendiment de la xarxa és degut al protocol Ethernet, en el que contra més pugem la mida de les dades a transmetre més augmenta l'ample de banda efectiu, ja que els paquets contenen més proporció de dades d'informació que dades de control.

Capítol 8

8 Integració StarSs amb NANOS++

En aquesta secció es tracta la integració del model de programació StarSs en l'arquitectura ARMv7 amb l'ajut de la llibreria de temps d'execució NANOS++.

El model de programació StarSs es basa en fer un graf de dependències sobre totes les tasques a paral·lelitzar d'un algorisme en temps de compilació per després decidir en temps d'execució quines es poden executar sense alterar que el resultat final sigui el mateix que el de el codi sèrie. La qüestió està en que el temps que perdem recorrent el graf en temps d'execució sigui un temps prou petit com per que compensi el guany de poder executar tasques que no podríem executar tan aviat si tinguéssim un planificació estàtica o una planificació dinàmica en profunditat d'un bucle, que és a grans trets el que té OpenMP 2.0.

StarSs està enfocat a nivell intra-node però també té suport inter-node per la qual cosa existeixen varis submodels.

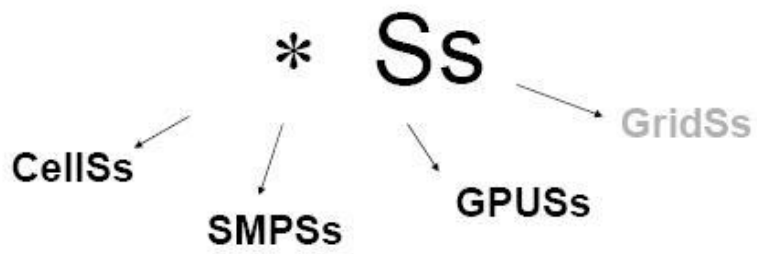

Figura 8.1: Diferents submodels de StarSs

- CellSs: Versió per a Cell.
- SMPSs: Versió per multicores i multiprocessadors simètrics, (SMP), com el processador usat en aquest projecte.
- GPUSs: Versió per GPUs.
- GridSs: Versió antiga per a grids, (inter-node). La versió nova es diu CompSs.

Per aconseguir el suport per l'arquitectura ARMv7, he implementat en codi assemblador els canvis de context per passar d'executar una tasca a executar-ne una altre, en la llibreria NANOS++.

8.1 Instal·lació de l'entorn NANOS++

- Anem a la web http://nanos.ac.upc.edu/content /nanos-environement-distribution
- Ens descarreguem el paquet Nanos environment distribution V0.1, [27*], [28*].
- És possible que haguem de modificar el fitxer configure.ac del directori /bsc-mcc-nanox-src/nanox-06a.tar.bz2/configure.ac per tal de que el autoconf sàpiga quin tipus de sistema operatiu ha de prendre com a objectiu.
    - Afegir a la linia 106 "OS=unix-os".
- Si tenim una versió anterior al Gcc i G++ 4.4.1 la desinstal·lem ja que les versions anteriors no linken bé objectes compartits que contene primitives atòmiques.
    - sudo apt-get –purge remove gcc g++
- Instal·lem la versió 4.4.1 del Gcc i el G++
    - sudo vim /etc/apt/sources.list
    - Canviem jaunty per karmic.
    - sudo apt-get update
    - sudo apt-get install gcc g++
- Instal·lem el processador de macros de llenguatges m4.
- Instal·lem el make si no el tenim.
    - sudo apt-get install make
- Instal·lem l'autoconf.
    - sudo apt-get install autoconf
- Posem la data del sistema a la data actual, per exemple:
    - sudo date -s "2 OCT 2006 18:00:00"
- Declarem la variable d'entorn TARGET apuntant al lloc on voldrem insta·lar l'entorn
    - ./install.sh $TARGET
    - export PATH=$PATH:/home/clusty/UtilitatsHPC/bsc-mcc-nanox-src/bin
    - sudo apt-get install libtool
    - sudo apt-get install bison
    - sudo apt-get install flex
    - sudo apt-get install gperf
    - sudo ap-get install git-core
- Instal·lem NANOS.
    - git clone git://nanos.ac.upc.edu/nanox
    - cd nanox

    - autoreconf -f -i
    - ./configure --disable-instrumentation --prefix=XXXX
    - make install
- Instal·lem Mercurium.
    - git clone git://nanos.ac.upc.edu/mcxx
    - cd mcxx
    - autoreconf -f -i
    - ./configure –enable-ompss --prefix=XXXX --with-nanox=XXXX
    - make install
- Comprovem que el Nanos funciona; al directori de tests de Nanos++ fer:
    - make check

8.2 Compilació amb SSCC

Si el codi conté MPI aquest és un exemple:

- sscc –l/usr/include/mpich2 –L/usr/lib –lmpi –o LUB LUB.c
- sscc –l/usr/include/mpich2 –L/usr/lib –lmpi -DCHECK_RESULT –o LUB LUB.c

O si el el codi no conté MPI, simplement:

- sscc –o LUB LUB.c

8.3 Estudi del canvi de context d'ARM a nivell de codi màquina.

8.3.1 Els registres i el seu correcte ús

En una arquitectura ARMv7 tenim els següents registres de 4 bytes a cada nucli.

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Taula 8.1: Registres en l'arquitectura ARMv7

A més a més per a cada coprocesador VFP, en el Cortex-A9 tenim els següents registres.

32 registres de presició simple, (4 bytes); s0-s31, que poden ser accedits també com a 16 registres de precisió doble; d0-d15, fent overlaping, (d0=s0-s1, etc.).

VFP-v3, afegeix 16 registres més de doble precisió sense opció a ser usats com a registres de simple precisió; d16-d31.

- Subrutina cridadora: és la subrutina que crida a una altre subrutina, en anglès és la "caller subroutine".

- Subrutina cridada: és la subrutina cridada per una altre subrutina, en anglès és la “callee subroutine”.
- Regsitres caller-save: són els registres que ha de salvar la subrutina que crida.
- Registres callee-save: són els registres que ha salvar la subrutina cridada.
    - r9.
    - r10-r11.
    - s16-s31

Els registres s16-s31 han de ser preservats a través de les crides a subrutines.

Els registres s0-s15 no necessiten ser preservats a través de les crides a subrutines.

8.3.2 El mecanisme de salt i retorn de subrutines

8.3.2.1 Pas de paràmetres

Els registres r0-r3 són usats per passar paràmetres entres subrutines i per retornar valors d’una funció.

Els registres s0-s15 poden ser usats per passar arguments o retornar resultats segons els diversos estàndards de crides a procediments d’ARM, [58*], [59*].

La política per passar els arguments és la següent:

- Si l’argument és un tipus compost del que no se’n pot saber la seva mida en temps de compilació, és copiat a memòria i l’argument és reemplaçat per un punter a la còpia en la pila.
- Si l’argument és més petit de 4 bytes, s’extè el seu signe fins a 4 bytes. Si l’argument és de mitja precisió es converteix a simple precisió.
- Si l’argument és candidat a ser passat per un registre d’un cooprocessador es passa per un d’aquests registres. Això només passa en variants del estàndard de crides a procediments. En cas que sigui així, s’utilitzen els registres s0-s15 els quals no necessiten ser preservats en les crides a subrutines. La política diu que per a cada argument candidat s’escolleix el conjunt de registres que comença pel registre de nombre més baix fins a passar tot l’argument per registres o sino queda espai en els registres per la pila, ja sigui a amb part de l’argument passat per registres o tot passat per la pila perquè no tinguem cap registre cooprocessador d’entrada.
- Si l’argument és un tipus compost i la seva mida no és múltiple de 4 bytes, la seva mida és arrodonida al múltiple de 4 més proper.
- Quan s’ens acaben els registres r0-r4 els arguments es passen per la pila, tenint un canvi de context més lent.

8.3.2.2 Salt i retorn

Abans de saltar a la direcció de memòria de programa on comença la subrutina cridada es copia la direcció següent del PC, (PC+4), el link register, (r14), per tal de saber quina és la direcció següent al retornar de la subrutina.

Retornar a la subrutina cridadora es pot fer de varies maneres. Es pot fer usant un MOV PC LR, la qual copiarà el link register al program counter, fent que la següent instrucció a executar sigui la guardada abans de saltar a la subrutina cridada. Una altre manera és fer un push de LR a la pila per posteriorment fer un pop a PC, la qual cosa copia el valor guardat de LR a PC al fer el pop. O per últim i el que usa el compilador, es pot usar la instrucció BX LR que produiex el mateix efecte.

En el nostre cas, com que sempre tenim la possibilitat d'haver de cridar a a més d'una subrutina imbricada, haurem de fer STMED r13!, {r0-r3, r14} al principi de cada subrutina i un LDMED r13!, {r0-r3, r14} al final de cada subrutina just abans de fer el MOV PC,LR, per tal de no perdre els valors d'aquests registres a través dels nivells d'imbricació de crides. Amb la qual cosa aquests registres que en principi no eren calle-save ara si ho són.

8.4 Implementació del canvi de context a ARM per a NANOS++

8.4.1 Algorisme

El codi es troba en l'annex, secció "codis de programa usats", codi 12.

8.4.2 Preparació del paquet NANOS++ per l'arquitectura ARMv7

- Modifcar el fitxer makefile.am.
- Modificar el fitxer configure.ac.
- Recompilar i reinstal·lar NANOS++.

8.4.3 Esquema dels blocs d'activació

Per a la implementació del canvi de context prenc la pila com a pila full descending, en la que els elements s'apilen de direcció més gran a més petita i on el stack pointer apunta a l'últim element apilat.

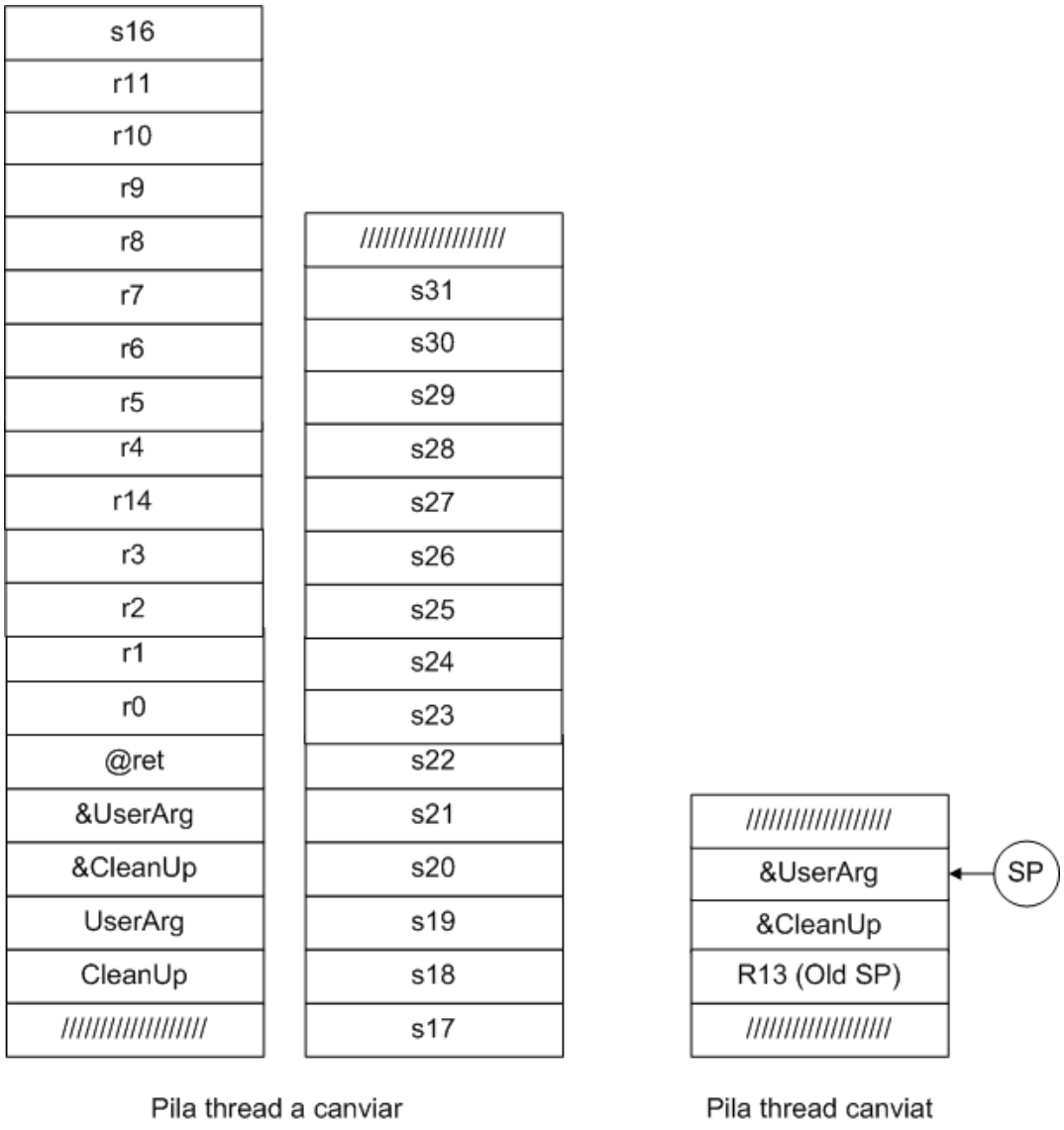

Figura 8.2: Blocs d'activació en l'arquitectura ARMv7

La rutina de canvi de context salva l'estat dels registres caller-save, canvia el stack pointer a l'espai de pila pel nou thread, li passa el seu stack pointer i 2 paràmetres i finalment fa crida al mètode del nou thread. A partir d'aquí, si el thread canviat volgués cridar a un altre thread es repetiria el procediment de salvar els registres caleer-save.

Capítol 9

9 Mesures de consum i temperatura

9.1 Mesures de consum

Aquí es tracta tot el referent al consum de la placa i es fan mesures de consum.

9.1.1 Consum del conjunt

Per calcular el consum en Watts del conjunt del components de la placa cal connectar un amperímetre en sèrie entre la sortida del transformador i l'entrada de la placa. Sabent que el voltatge que subministra el transformador és de corrent continua a 15 V, podem saber la potència consumida en Watts aplicant P=V*I.

En totes les mesures he tingut en compte els consums quan els nuclis treballen a 1 GHz i quan treballen a 618 MHz. Com ja hem vist en capítols anteriors el SoC es comporta de manera que quan es demana tot el rendiment d'un nucli es comença executant amb 1GHz de freqüència fins que el sistema d'administració d'energia decideix baixar la freqüència a una freqüència determinada podent tornar a 1 GHz durant la mateixa execució si el sistema d'administració d'energia ho decideix.

9.1.1.1 Consum de la placa en estat de repòs

Per calcular el consum de la placa en estat de repòs m'he basat en fer la mitjana del que consumeix en el temps, ja que el consum oscil·la permanentment en 0,001-0,002 A amb alguns pics d'Amperes que són deguts a demons del SO.

Estat dels components de la placa	Hz processador	Amperes consumits	Watts consumits
Ethernet desconnectat	1 nucli, 40 Hz constants (pics de 75 Hz)	0,177-0,178 A	2,655-2,670 W
Ethernet connectat	1 nucli, 40 Hz constants (pics de 75 Hz)	0,196-0,198 A	2,94-2,97 W

Taula 9.1: Resultats del consum de la placa en estat de repòs

Del que traiem que el Ethernet connectat consumeix una mitja de 0,29 Watts.

Totes les mesures que he fet posteriorment estan fetes amb el cable Ethernet connectat.

9.1.1.2 Consum de la placa processant amb un nucli

Algorisme	Amperes consumits a 1000 MHz	Amperes consumits a 618MHz	Watts mitjos consumits a 1000 MHz	Watts mitjos consumits a 618 MHz
Multiplicació de matrius	0,290-0,285	0,241-0,239	4,31	3,6
Producte vectorial	0,275-0,273	0,229-0,227	4,11	3,4
LU	0,282-0,279	0,237-0,235	4,21	3,54
Cholesky	0,283-0,281	0,237-0,235	4,23	3,54

Taula 9.2: Resultats del consum de la placa processant amb un nucli

Veiem com els consums varien considerablement segons la freqüència del nucli. La mitjana de diferències de consum dels diferents algorismes per a execucions usant un sol nucli és de 0,70 Watts; si tenim en compte que el consum del processador declarat per ARM en freqüència de 800 MHz és de 0,5 Watts usant els dos nuclis, ens donem compte de que el conjunt està consumint més que el que consumeix el processador en estat de rendiment declarat només pel fet d'incrementar la freqüència en 200 MHz. Aquest consum està lligat a major lligat a major consum del processador com també està lligat a major demanda a la memòria RAM.

Ara calculem quina diferencia hi ha de potencia entre tenir el processador amb freqüència i càrrega de treball de repòs i tenir-lo amb la freqüència més elevada i la càrrega de treball al màxim, (95%-100%).

Mitjana de potències a 1000 MHz	4,22 W
Mitjana de potències a 618 MHz	3,52 W
Mitjana de potències prenent que el 25% del temps treballa a 1000 MHz i el 75% del temps treballa a 618 MHz	3,70 W
Diferència de potències en estat de repòs a estat d'execució	0,74 W

Taula 9.3: Comparacions de consum segons la freqüència de treball

Tenim que la diferència de passar de repòs a executar consumeix una mitja de 0,74 W

9.1.1.3 Consum de la placa processant amb dos nuclis

Algorisme	Amperes consumits a 1000 MHz	Amperes consumits a 618MHz	Watts consumits a 1000 MHz	Watts consumits a 618 MHz
Multiplicació de matrius	0,313-0,321	0,281-0,278	4,76	4,19
Producte vectorial	0,299-0,297	0,239-0,237	4,47	3,57
LU	0,317-0,315	0,255-0,252	4,74	3,80
Cholesky	0,316-0,312	0,253-0,250	4,71	3,77

Taula 9.4: Resultats del consum de la placa processant amb dos nuclis

9.1.1.4 Consum de la placa processant amb dos nuclis i transmetent per la xarxa

Estat dels components de la placa	Amperes consumits a 1000 MHz	Amperes consumits a 618MHz	Watts consumits a 1000 MHz	Watts consumits a 618 MHz
LU	0,292-0,283	0,236-0,223	4,31	3,44
Cholesky	0,297-0,289	0,238-0,233	4,40	3,53
Ping-pong	0,272-0,259	0,232-0,229	3,98	3,46

Taula 9.5: Resultats del consum de la placa processant amb dos nuclis i transmetent per la xarxa

9.1.2 Cosum dels components

Per saber el consum dels components hauria de poder posar un voltímetre i un amperímetre a l'entrada dels components, com això és impossible amb les eines que tinc he optat per deduir aproximadament quin consum s'estan emportant el components que consumeixen.

Consum del conjunt treballant a 817 MHz a càrrega del processador del 100%:	3,52 W
Consum del Cortex-A9 declarat per ARM treballant a 618 MHz	0,5 W
Consum del Ethernet només pel fet d'estar endollat	0,29 W
Consum del USB mínim del USB, (5V 100mA-500mA)	0,5 W

Consum aproximat de la memòria RAM	1,8 W
Consum de la resta de components (resta del SoC, components placa i busos)	0,43 W

Taula 9.6: Especulació del consum dels components de la placa

Ens queda que la resta de components consumeixen 2,23 W.

9.2 Mesures de temperatura

Per tal de saber més sobre el comportament del sistema d'administració d'energia i del que podrien necessitar per tal de dissipar la calor del SoC he fet algunes mesures de temperatura.

El instrument que he usat per fer les mesures és un termòmetre de rajos infrarojos que mesura dels –33 als +220°C; d'aquesta manera el termòmetre no toca el processador sinó que mesura el rajos infrarojos que desprèn aquest i els mesura.

Figura 9.1: Termòmetre de rajos infrarojos usat

Les condicions de mesura han estat de 23°C de temperatura d'ambient.

9.2.1 Mesures de temperatura del SoC en estat de repós

Esdeveniments en la placa	Freqüència de treball en MHz	Graus Celsius
Placa encesa al cap de 20 minuts sense haver calculat res	40-50	40,4
Placa encesa al cap d'una hora i mitja havent estat calculant	40-50	41,3

Taula 9.7: Resultats de les mesures de temperatura del SoC en estat de repòs

La temperatura que esperem tenir en la placa en estat de repòs estaria entre els 41 i 42 °C. Si configurem la placa per que treballi amb tots els seus components a màxima freqüència, és a dir el processador a 1000 MHz en estat de repòs. La temperatura en estat de repòs s'eleva a 44,4 °C.

9.2.2 Mesures de temperatura del SoC usant un nucli

Esdeveniments en la placa	Freqüència de treball en MHz	Graus Celsius
Placa calculant un producte escalar amb OpenMP amb 1 thread	1000 MHz	47,3
Placa calculant un producte escalar amb OpenMP amb 1 thread	618 MHz	45

Taula 9.8: Resultats de les mesures de temperatura del SoC usant un nucli

En les execucions usant un sol nucli la freqüència del processador va alternant entre 1000 MHz i 618 MHz. El comportament és que el processador treballa a 1000 MHz i la temperatura va augmentat, quan arriba a uns 47,3 °C el sistema d'administració d'energia baixa la freqüència a 618 MHz fins que la temperatura baixa uns 45 °C i torna a pujar la freqüència a 1000 MHz.

Configurant la placa per fer treballar tots els dispositius al màxim de freqüència amb la comanda tegrastars –max, no m'ha aportat cap benefici. El comportament ha estat el mateix, i les temperatures han estat en el interval de 47,1 a 44,9 °C.

9.2.3 Mesures de temperatura del SoC usant dos nuclis

Esdeveniments en la placa	Frequència de treball en MHz	Graus Celsius
Placa calculant un producte escalar amb OpenMP amb 4	1000 MHz	46,8

threads		
Placa calculant un producte escalar amb OpenMP amb 4 threads	618 MHz	44,4
Placa calculant una multiplicació de matrius amb OpenMP amb 4 threads	618	46,9

Taula 9.9: Resultats de les mesures de temperatura del SoC usant dos nuclis

Amb dos nuclis treballant les temperatures han estat una mica més baixes que amb un nucli, es possible que sigui degut a que els dos nuclis passin algun temps aturats degut a esperes per coherència.

L'execució de la multiplicació de matrius és un cas que ens revela informació important. Córrer l'algorisme a 618 MHz dissipa tanta calor com córrer l'algorisme de producte escalar a 1000 MHz. El que vol dir que l'algorisme de multiplicació de matrius fa un ús més intensiu del pipeline dels nuclis i que per tant al escalfar més el processador el sistema d'administració d'energia baixa la freqüència fins a tenir una temperatura prudent.

En aquest experiment també vaig provar de posar a treballar tots els elements a freqüència màxima amb la comanda, tegrastats –max. El rendiment va ser pitjor perquè el sistema d'administració d'energia un cop baixada la freqüència a 618 MHz no la tornava a pujar encara que la temperatura arribés a baixar fins a 45,3 °C.

Capítol 10

10 Conclusions

El Cortex-A9 de doble nucli en la versió per optimitzar consum, condicionat per estar en la placa de desenvolupament, ha rendit per sota de la resta de processadors a les proves de benchmarks generals però, a canvi ha donat un rendiment per Watt molt superior. Tenint en compte que ha estat desenvolupat per dispositius mòbils, ha estat sorprenent que es defenses prou bé amb algorismes de computació amb coma de doble precisió.

En les seccions de verificació del funcionament, compilació òptima i d'algorismes propis, he pogut comprovar que si els algorismes estan ben programats l'arquitectura pot escalar segons el nombre de threads, ja sigui amb pthreads o amb OpenMP.

El clúster que he muntat està bastant limitat per tenir una xarxa d'interconnexió d'alta latència i poc ample de banda. No obstant, he aconseguit, usant MPI i OpenMP, tenir speed-ups superlineals quan la mida de l'entrada de les dades és prou grossa.

Un altre aspecte, però que no varia la diferència entre executar el codi sèrie del paral·lel, ha estat la memòria RAM que ha fet rendir els processadors per sota del que podrien rendir tot i tenir jerarquia de memòria.

Per poder haver profunditzat més en les conclusions de cada apartat m'he vist afectat perquè l'arquitectura és tan recent que encara no hi ha suport per comptatge de events. És per això que no he pogut usar eines com Oprofile, Gprof, Paraver, Dimemas, etc.

Pel que respecta les mesures de consum i temperatura he pogut comprovar com ARM declara que el processador treballa a 0,8 GHz mentre que la realitat és que el processador dins del SoC Tegra 2 comença executant a 1 GHz per baixar a 618 MHz i tornar a pujar quan la temperatura baixa suficientment; aquest fet és perfectament comprensible ja que el SoC està destinat a un mercat on s'executen aplicacions que no necessitaran gaire potencial de càlcul però l'usuari vol velocitat en la càrrega d'aplicacions. Escalfant i refredant artificialment el processador vaig poder esbrinar que el sistema d'administració d'energia es guia per un sensor de temperatura i no per exemple per un amperímetre amb algun temporitzador. També he pogut observar que alguns algorísmes no exigien tant al processador i per tant no l'escalfaven tant, com a conseqüència el processador no baixava tant de MHz i per tant tenien una mitja de consum major.

Resumint, ens queda que el SoC Tegra 2 no és una mala aproximació per a la computació d'altes prestacions però pesa molt que hagi estat pensat per dispositius mòbils i això fa que no sigui realment una opció. El que si que és molt interessant és el Cortex-A9 i la seva arquitectura enfocada al baix consum, basada en factors com el seu pipeline de poques etapes i el seu joc d'instruccions simple. Tenint en compte els

processadors que hem vist anunciats pel 2011 i 2013 i que els seus consums no s'espera que baixin molt, seria una bona idea estar a la espera del nou Cortex-A15 anunciat per ARM. Podria satisfer el potencial de càlcul per node que es necessita per la computació d'altres prestacions d'avui en dia amb un consum molt per sota de la competència.

Capítol 11

11 Planificació seguida

11.1 Tasques

Nom de la tasca	Descripció de la tasca
Instal·lació de Linux L4T a la placa	Instal·lació de un Ubuntu compilat per el ARM Cortex-A9.
Comprovació del funcionament	Verificació del funcionament de la placa i del seu suport per a córrer més d'un procés i/o fil d'execució repartint-los entre els seus nuclis.
Compilació òptima	Trobar la combinació de flags òptima per executar codis sèrie i paral·lelitzats en la plataforma.
Benchmarks	Execució i valoració de diferents benchmarks per poder comparar el rendiment del processador i memòria amb altres.
Mesures amb Pthreads i OpenMP	Execució de codis triats per comprovar-ne els seu rendiment amb Pthreads i OpenMP.
Muntatge del clúster	Muntatge d'un clúster beowulf usant les plaques de desenvolupament per provar el MPI en diversos nodes.
Paralel·lització de factoritzacions de matrius	Programació de parale·litzacions de factoritzacions LU i Cholesky per al clúster en concret.
Mesures amb MPI + OpenMP	Execució de les paralel·litzacions en el clúster i conclusions.
Mesures de temperatura i consum	Mesures de temperatura i consum d'un node del clúster en diversos estats.
Estudi de traces a ARM	Cerca d'informació per fer traces d'execucions en el clúster.
Rutina en assemblador NANOS++	Implementació de canvi de context a NANOS++ en llenguatge assemblador.

Taula 11.1: Descripció de les tasques del projecte

11.2 Diagrama de Gantt

Nombre	Fecha d...	Fecha d...
Inscripció PFC	1/07/10	3/07/10
Instal·lació Linux L4T	1/07/10	10/07/10
Verficació funcionament placa	1/07/10	31/07/10
Part introductoria memòria	12/07/10	4/09/10
Informació sobre el SoC	2/08/10	4/09/10
Anex memòria	26/07/10	4/09/10
Compilació òptima	26/07/10	14/08/10
Benchmarks	5/08/10	20/08/10
Mesures Pthreads	18/08/10	18/09/10
Mesures OpenMP	18/08/10	18/09/10
Muntatge del clúster	27/08/10	4/09/10
Afinament MPI	6/09/10	18/09/10
Factoritzacions matrius	13/09/10	29/09/10
Mesures MPI	1/10/10	16/10/10
Mesures de temperatura i consum	15/10/10	2/11/10
Estudi de traces en ARM	3/11/10	9/11/10
Revisió de la memòria	10/11/10	10/01/11
Rutines Nanos++	26/11/10	14/01/11
Preparació defensa projecte	10/01/11	19/01/11
Límit registre memòries	19/01/11	20/01/11
Limit lectura PFC	26/01/11	27/01/11

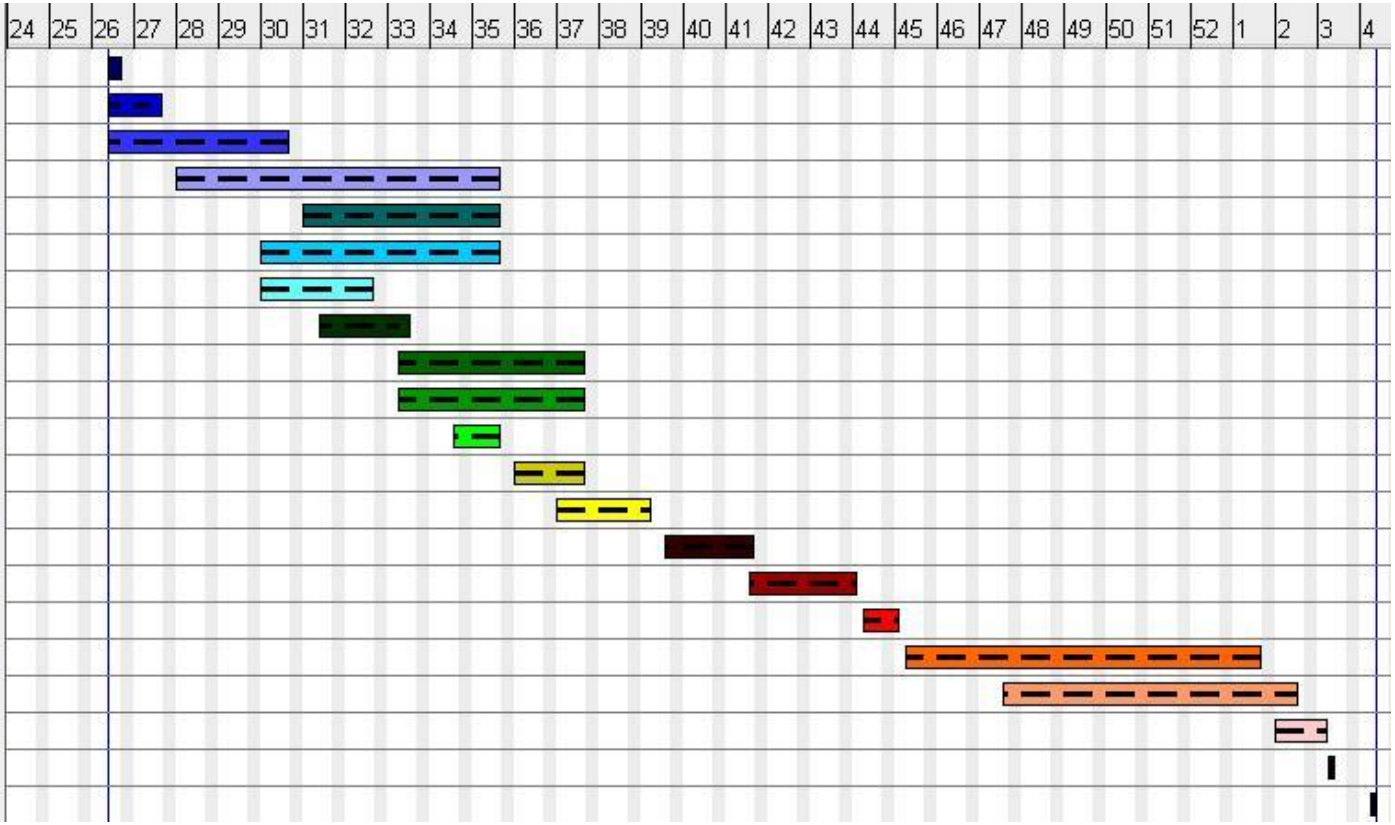

Figura 11.1: Diagrama de gantt de les tasques del projecte

Capítol 12

12 Valoració econòmica

En quant a la valoració econòmica del projecte, exposo quin és el preu dels components hardware i el software utilitzat. En cas de que el software o hardware utilitzat hagi estat proporcionat per el centre o altres fonts també es suma el seu preu.

12.1 Hardware

Nom del hardware	**Preu**
Placa de desenvolupament Nvidia Tegra 200 series x2	620 €
Memòria USB 8 GB x2	50 €
Cable Ethernet x3	30 €
Router x1	50 €
Tester x1	35 €
Termòmetre de rajos infrarojos x1	35 €
Preu Total	820 €

Taula 12.1: Cost del hardware usat en el projecte

12.2 Software

Nom del software	**Preu**
Compilador Gcc	Gratuït
NetPipe	Gratuït
MPICH 2	Gratuït
Dhrystone Benchmark	Gratuït
Nbench Benchmark	Gratuït
Stream Benchmark	Gratuït
Openssh	Gratuït
NFS	Gratuït
Ubuntu L4T	Gratuït

Mcc-nanos++	Gratuït
Preu Total	0 €

Taula 12.2: Cost del software usat en el projecte

12.3 Tasques

En aquest projecte he hagut de desenvolupar els rols següents:

- Analista: dissenya i planifica les tasques de programació.
- Programador: implementa i realitza les tasques de programació.
- Tècnic de sistemes i xarxes: munta les infraestructures per desenvolupar i/o usar el software.
- Investigador: en aquest cas, experimenta i treu conclusions sobre les probes d'un software que corre en un hardware en concret.

Rol	Acrònim	Preu per hora
Analista	AN	50 €
Programador	PROG	40 €
Tècnic de sistemes i xarxes	TSX	40 €
Investigador	INV	60 €

Taula 12.3: Rols i preus dels rols desenvolupants en el projecte

Nom de la tasca	**Rol**	**Hores dedicades**	**Preu**
Instal·lació de Linux L4T a la placa	TSX	50	2000
Comprovació del funcionament	PROG	5	200
Compilació òptima	PROG	25	1000
Benchmarks i mesures amb threads	INV	75	4500
Muntatge del clúster	TSX	75	3000
Paralel·lització de factoritzacions de matrius	AN + PROG	75	3375
Mesures amb MPI + OpenMP	INV	50	3000
Mesures de temperatura i consum	INV	50	3000
Estudi de traces a ARM	AN	25	1250
Rutina en assemblador NANOS++	AN + PROG	75	3375
Esctritura de la memòria	TSX + INV	200	10000

Preu i hores totals		705 hores	34700 €

Taula 12.4: Cost del projecte

13 Annex

13.1 Codis de programes usats

13.1.1 Algorisme de creació de processos

```
#define NFORKS 50000
void do_nothing() {
int i; i= 0;}

int main() {
int pid, j, status;
for (j=0; j<NFORKS; j++) {

  if ((pid = fork()) < 0 ) {
    printf ("fork failed with error code= %d\n", pid);
    exit(0); }

  else if (pid ==0) { do_nothing(); exit(0); }

  else { waitpid(pid, status, 0);}}
  return 0;}
```

Codi 1: Algorisme de creació de processos

13.1.2 Algorisme de creació de threads

```
#define NTHREADS 50000

void *do_nothing(void *threadArgs){
    int i;
    i= 0; pthread_exit(NULL);}

int main(int argc, char *argv[]){
    pthread_t thread[NTHREADS];
    int rc, t;

    for(t=0;t < NTHREADS;t++) {
        rc = pthread_create(&thread[t], NULL, do_nothing, NULL);
        if (rc) {
            printf("ERROR; return code from pthread create()is %d\n", rc); exit(-1);}
        rc = pthread_join(thread[t], NULL);
        if (rc) {
            printf("ERROR; return code from pthread join()is %d\n", rc); exit(-1);}
    }
    pthread_exit(NULL);
    return(0);
}
```

Codi 2: Algorisme de creació de threads

13.1.3 Algorisme de producte escalar

```
void* dotprod(void){

   int start, end, i, j;
   double mysum, *x, *y;

   start=0;
   end = dotstr.veclen;
   x = dotstr.a; y = dotstr.b;

   mysum = 0;
   for (j=0; j<1000; j++){
     #pragma omp parallel for schedule (runtime) private(i) reduction
(+:mysum)
     for (i=start; i<end ; i++){
       mysum += (x[i] * y[i]);}}
   dotstr.sum = mysum;

   return ;
}
```

Codi 3: Algorisme de producte escalar

13.1.4 Algorisme de multiplicació de matrius

```
void* mult1(void *t){

  int i,j,k,l;
  int sum;

  for ( i=0 ; i < N; i++ ){ //O(N^3)
    for ( j=(int)t ; j < N; j+=NTHREADS ){
    for (l=0; l<N;l++){
     C[i][j]+=A[i][l]*B[l][j];
    }
    }
  }
}
```

Codi 4: Algorisme de multiplicació de matrius

13.1.5 Algorisme de producte escalar amb paralel·lització OpenMP

```
void* dotprod(void *t){
   double mysum, *x, *y;
   start=(int)t*((dotstr.veclen)/NTHREADS);
   end = start+(dotstr.veclen)/NTHREADS;

   if((int)t==NTHREADS-1){
    if((dotstr.veclen)%NTHREADS!=0){
        end=end+(dotstr.veclen)%NTHREADS;}}
   x = dotstr.a; y = dotstr.b; mysum = 0;

   for (j=0; j<1000; j++){
     for (i=start; i<end ; i++) {
        mysum += (x[i] * y[i]); }}

    pthread_mutex_lock(&mut); dotstr.sum += mysum;
    pthread_mutex_unlock(&mut);
   return;}
```

Codi 5: Algorisme de producte escalar amb paralel·lització OpenMP

13.1.6 Algorisme de multiplicació de matrius amb paralel·lització OpenMP

```
void mult1 ()
{
  int i,j,k;
  int sum;

  #pragma omp  parallel for private (i,j,k) schedule (static)
  for ( i=0 ; i < n; i++ ) //O(n^3)
  {
    for ( j=0 ; j < n; j++ )
    {
      for ( k=0 ; k < n ; k++ )
         C[i][j]+= A[i][k]*B[k][j];
    }
  }
}
```

Codi 6: Algorisme de multiplicació de matrius amb paralel·lització OpenMP

13.1.7 Algorisme de multiplicació de factorització de matrius LU

```
int main(int argc, char *argv[]){

      for (kk=0; kk<N; kk+=B) {

     if(N-kk<NUM_THR){
         B=N-kk;
         lu0p(kk,B);
     }
     else{ B=(((N-kk)/NUM_THR)+1);

     lu0(kk,B);

     #pragma omp parallel for schedule(runtime) private(jj)
          for (jj=kk+B; jj<N; jj+=B){
              fwd(kk, jj,B);
     }
     #pragma omp parallel for schedule(runtime) private(ii, jj)
          for (ii=kk+B; ii<N; ii+=B) {
             bdiv (ii, kk,B);
             for (jj=kk+B; jj<N; jj+=B) {
                bmod(ii, jj, kk,B);
         }
             }
         }
          }
       return 0;
}
```

Codi 7: Algorisme de multiplicació de factorització de matrius LU

13.1.8 Algorisme seqüencial de factorització Cholesky

```
for (k=0; k<N ; k++){
      A[k][k]= sqrt(A[k][k]);

      #pragma omp parallel for schedule(runtime) private(j)
      for (j=k+1; j<N; j++){
         A[k][j] = A[k][j] / A[k][k];
      }
      #pragma omp parallel for schedule(runtime) private(i,j)
      for(i=k+1; i<N; i++){
         for(j=i; j<N; j++){
            A[i][j]= A[i][j] - A[k][i] * A[k][j];
          }
      }
}
```

Codi 8: Algorisme seqüencial de factorització Cholesky

13.1.9 Algorisme de factorització LU amb blocking i MPI

```
int main(int argc, char* argv[]){

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD, &howmany);
   MPI_Comm_rank(MPI_COMM_WORLD, &whoAmI);

      NB = N/B;
    if(N%B!=0) NB++;
    for(b=whoAmI;b<NB;b+=howmany)
        nfiles+=MIN(B,B-((B*b)-N));

      A = malloc(nfiles*sizeof(double *));
    for(ii=0;ii<nfiles;ii++)
        A[ii]=malloc(N*sizeof(double));

      genmat();
      t_start=usecs();
    NB = (N+B-1)/B;

    for (kk=0; kk<NB; kk++) {
    if(kk%howmany==whoAmI){
        lu0(kk);
        for(i=0; i<MIN(B, N-kk*B); i++){
            for(j=0;j<MIN(B, N-kk*B);j++){
                lu0pack[i][j]=A[((kk/howmany)*B)+i][kk*B+j];
    }}}

    MPI_Bcast(&lu0pack[0][0],B*B,MPI_DOUBLE,kk%howmany,MPI_COMM_WORLD);
    for(ii=kk+1;ii<NB;ii++){
        if(ii%howmany==whoAmI){
        bdiv(ii,kk);
        }}
    for(jj=kk+1;jj<NB;jj++){
        if(kk%howmany==whoAmI){
        fwd(kk,jj);
        for(i=0; i<MIN(B, N-kk*B); i++){
                        for(j=0;j<MIN(B, N-jj*B);j++){
            fwdpack[i][j]=A[((kk/howmany)*B)+i][jj*B+j];
      }}}
        MPI_Bcast(&fwdpack[0][0],B*B,MPI_DOUBLE,kk%howmany,MPI_COMM_WORLD);
        for(ii=kk+1;ii<NB;ii++){
            if(ii%howmany==whoAmI){
            bmod(ii,jj,kk);
    }}}}
      t_end=usecs();
      time = ((double)(t_end-t_start))/1000000;
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_Finalize();
}
```

Codi 9: Algorisme de factorització LU amb blocking i MPI

13.1.10 Algorisme de factorització Cholesky amb blocking i MPI

```
int main(int argc, char* argv[]){
   long t_start,t_end;
   double time;
   int NB, b;
   int ii, jj, kk;
   int nfiles=0;
   int i,j;

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD, &howmany);
   MPI_Comm_rank(MPI_COMM_WORLD, &whoAmI);

      NB = N/B;
    if(N%B!=0) NB++;
    for(b=whoAmI;b<NB;b+=howmany)
    nfiles+=MIN(B,B-((B*b)-N));

      A = malloc(nfiles*sizeof(double *));

    for(ii=0;ii<nfiles;ii++)
        A[ii]=malloc(N*sizeof(double));

      genmat();
      t_start=usecs();
    NB = (N+B-1)/B;
    for (kk=0; kk<NB; kk++) {
    if(kk%howmany==whoAmI){
        chole(kk);}

    for(jj=kk+1;jj<NB;jj++){
        if(kk%howmany==whoAmI){
        bdiv(kk,jj);
        for(i=0; i<MIN(B, N-kk*B); i++){
                       for(j=0;j<MIN(B, N-jj*B);j++){
                       bdivpack[i][j]=A[((kk/howmany)*B)+i][jj*B+j];
      }}}
    MPI_Bcast(&bdivpack[0][0],B*B,MPI_DOUBLE,kk%howmany,MPI_COMM_WORLD);

        for(ii=kk+1;ii<=jj;ii++){
            if(ii%howmany==whoAmI){
            bmod(ii,jj,kk);
            }}}
        }
      t_end=usecs();
      time = ((double)(t_end-t_start))/1000000;
    MPI_Barrier(MPI_COMM_WORLD);
    printf("proces %d, time to compute = %f\n", whoAmI, time);
    MPI_Finalize();
}
```

Codi 10: Algorisme de factorització Cholesky amb blocking i MPI

13.1.11 Algorísme de ping-pong

```
MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myproc);

   if (nprocs != 2) exit (1);
   other_proc = (myproc + 1) % 2;

   printf("Hello from %d of %d\n", myproc, nprocs);
   MPI_Barrier(MPI_COMM_WORLD);

/* Timer accuracy test */

   t0 = MPI_Wtime();
   t1 = MPI_Wtime();

while (t1 == t0) t1 = MPI_Wtime();

   if (myproc == 0)
      printf("Timer accuracy of ~%f usecs\n\n", (t1 - t0) * 1000000);

/* Communications between nodes
 *   - Blocking sends and recvs
 *   - No guarantee of prepost, so might pass through comm buffer
 */
   for (size = 8; size <= 1048576; size *= 2) {
      for (i = 0; i < size / 8; i++) {
         a[i] = (double) i;
         b[i] = 0.0;
      }
      last = size / 8 - 1;

      MPI_Barrier(MPI_COMM_WORLD);
      t0 = MPI_Wtime();

      if (myproc == 0) {

         MPI_Send(a, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD);
         MPI_Recv(b, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD,
&status);

      } else {

         MPI_Recv(b, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD,
&status);

         b[0] += 1.0;
         if (last != 0)
         b[last] += 1.0;

         MPI_Send(b, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD);

      }
      t1 = MPI_Wtime();
      time = 1.e6 * (t1 - t0);
      MPI_Barrier(MPI_COMM_WORLD);
```

```
      if ((b[0] != 1.0 || b[last] != last + 1)) {
         printf("ERROR - b[0] = %f b[%d] = %f\n", b[0], last, b[last]);
         exit (1);
      }

     for (i = 1; i < last - 1; i++)
         if (b[i] != (double) i)
            printf("ERROR - b[%d] = %f\n", i, b[i]);
      if (myproc == 0 && time > 0.000001) {
         printf(" %7d bytes took %9.0f usec (%8.3f MB/sec)\n",
                     size, time, 2.0 * size / time);
         if (2 * size / time > max_rate) max_rate = 2 * size / time;
         if (time / 2 < min_latency) min_latency = time / 2;
      } else if (myproc == 0) {
         printf(" %7d bytes took less than the timer accuracy\n", size);
      }
   }
   for (size = 8; size <= 1048576; size *= 2) {
      for (i = 0; i < size / 8; i++) {
         a[i] = (double) i;
         b[i] = 0.0;
      }
      last = size / 8 - 1;
      MPI_Barrier(MPI_COMM_WORLD);
      t0 = MPI_Wtime();

      if (myproc == 0) {
         MPI_Send(a, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD);
         MPI_Recv(b, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD,
&status);
      } else {

         MPI_Recv(b, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD,
&status);
         b[0] += 1.0;
         if (last != 0)
         b[last] += 1.0;
         MPI_Send(b, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD);
      }
      t1 = MPI_Wtime();
      time = 1.e6 * (t1 - t0);
      MPI_Barrier(MPI_COMM_WORLD);

      if ((b[0] != 1.0 || b[last] != last + 1)) {
         printf("ERROR - b[0] = %f b[%d] = %f\n", b[0], last, b[last]);
         exit (1);
      }
      for (i = 1; i < last - 1; i++)
         if (b[i] != (double) i)
            printf("ERROR - b[%d] = %f\n", i, b[i]);
      if (myproc == 0 && time > 0.000001) {
         printf(" %7d bytes took %9.0f usec (%8.3f MB/sec)\n",
                     size, time, 2.0 * size / time);
         if (2 * size / time > max_rate) max_rate = 2 * size / time;
         if (time / 2 < min_latency) min_latency = time / 2;
```

Codi 11: Algorísme de ping pong

13.1.12 Canvi de context de NANOS++ per arquitectura ARMv7

```
    .arch armv7-a
    .eabi_attribute 27, 3
    .fpu vfp
    .eabi_attribute 20, 1
    .eabi_attribute 21, 1
    .eabi_attribute 23, 3
    .eabi_attribute 24, 1
    .eabi_attribute 25, 1
    .eabi_attribute 26, 2
    .eabi_attribute 30, 6
    .eabi_attribute 18, 4
    .file    "switchStacks.c"
    .text
    .align    2
    .global    switchStacks
    .type    switchStacks, %function
switchStacks:
    stmed r13!, {r0-r3, r14}
    stmed r13!, {r4-r11}
    fstmfds sp!, {s16-s31}
    mov r0, r13
    add r13, r13, #128
    ldmed r13, {r13}
    push {r0}
    add r1, r0, #124
    push {r1}
    add  r1, r0, #120
    push {r1}
    add r1,  r0, #132
    ldmed r1, {r1}
    blx r1
    add r13, r13, #12
    fldmfds sp!, {s16-s31}
    ldmed r13!, {r4-r11}
    ldmed r13!, {r0-r3, r14}
    mov PC, LR
    .size    switchStacks, .-switchStacks
    .ident    "GCC: (Ubuntu 4.4.1-4ubuntu8) 4.4.1"
    .section    .note.GNU-stack,"",%progbits
```

Codi 12: Canvi de context de NANOS++ per arquitectura ARMv7

13.2 Monitorització

13.2.1 Instal·lació Paraver

Paraver és una eina que ens permet visualitzar traces de programes paral·lels. Per instal·lar-lo fem el següent:

- Anem a http://www.bsc.es/plantillaC.php?cat_id=625
- Descarreguem el paquet New Paraver,Linux-x86 32 bits i el descomprimim.
  - tar xvf wxparaver32.tar.gz
- Exportem la variable d'entorn PARAVER_HOME i executem.
  - export PARAVER_HOME=/home/clusty/pathfinsalacarpeta/wxparaver32
  - ./wxparaver

13.2.2 Instal·lació Extrae

- Ens descarreguem el codi font del lloc web:
  - http://www.bsc.es/plantillaC.php?cat_id=625
- Descomprimim el paquet i ens posicionem dins del directori on hem descomprimit.
  - ./configure
  - make.
  - make install
- Quan ens indiqui que no troba l'arxiu xml2-config hem d'instal·lar el paquet libxml2-dev
  - sudo apt-get install libxml2-dev

13.2.3 Instal·lació PAPI

- Anem al lloc web http://icl.cs.utk.edu/papi/software/ i ens descarreguem l'última versió de PAPI.
- Descomprimim el paquet.
- Configurem i instal·lem PAPI.
  - ./configure
  - make
  - make install

13.2.4 Com tracejar aplicacions

Configurar tot l'entorn per tracejar és una tasca elaborada, anem a veure com es fa, [14], [15], [16], [6*].

Fins ara hem instal·lat els tres components principals que són els paquets de MPICH 2, els codis font del paquet d'instrumentació Extrae 2.0 i l'aplicació per monitoritzar les traces, el Paraver. Suposarem que tenim els tres components desplegats en els següents paths:

- MPICH2: a cada node del clúster. /usr
- Extrae: a cada node del clúster:/usr/local
- Paraver al node on tenim entorn gràfic, no cal que sigui un node de còmput: /home/clusty/pathfinsalacarpeta/wxparaver32.

El primer de tot és configurar les variables d'entorn següents, és recomanable fer-ho amb un script amb comandes export.

- MPITRACE_HOME=/usr/local
  - Aquesta vaiable serveix per indicar on està instal·lat el tracejador de MPI i/o OpenMP.
- LD_LIBRARY_PATH=/usr/local/lib/
  - Serveix per
- LIBXML2_HOME=/usr/
  - Indica on està instal·lada la llibreria xml2 que hem de tenir per fer funcionar el conjunt.
- LD_PRELOAD=libmpitrace.so
  - En el cas de que estiguem tracejant sense usar dyninst, per exemple, usant PAPI, hem de dir quina és la llibreria que intercepta les crides a la llibreria MPI i/o OpenMP.
- MPTRACE_CONFIG_FILE=./mpitrace.xml
  - Hem de dir on està situat el fitxer que configura les opcions de la traça. Aquest fitxer l'explico més endavant en la memòria.
- export PAPI_HOME=/usr/local
  - Indicar el path on està instal·lada la utilitat per manegar el comptadors de processadors independentment de l'arquitectura PAPI.
- export MPI_HOME=/usr
  - Indica el path on estàn les llibreries i executables MPI.

A continuació veurem un exemple d'arxiu de configuració per a les traces. Cal dir que si no es vol usar arxiu de configuració també es poden usar variables d'entorn, alguns exemples són:

- export MPITRACE_ON=1
  - especifiquem que volem habilitar el tracejar MPI.

- MPI_TRACE_MPI_COUNTERS_ON=1
  - Si MPI ha de reportar els valors dels comptadors de rendiment.

```
<trace enabled="yes" home="/usr/local" initial-mode="detail" type="paraver" xml-parser-id="Id: xml-parse.c 260 2010-05-14 14:06:21Z gllort $">
	<mpi enabled="yes">
		<counters enabled="yes"/>
	</mpi>
	<pacx enabled="no">
		<counters enabled="yes"/>
	</pacx>
	<openmp enabled="yes">
		<locks enabled="no"/>
		<counters enabled="yes"/>
	</openmp>
	<callers enabled="yes">
		<mpi enabled="yes">1-3</mpi>
		<pacx enabled="no">1-3</pacx>
		<sampling enabled="no">1-5</sampling>
	</callers>
	<user-functions enabled="no" list="/home/bsc41/bsc41273/user-functions.dat">
		<max-depth enabled="no">3</max-depth>
		<counters enabled="yes"/>
	</user-functions>
	<counters enabled="yes">
		<cpu enabled="yes" starting-set-distribution="1">
			<set enabled="yes" domain="all" changeat-globalops="5">
			PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM
			<sampling enabled="no" frequency="100000000">
			PAPI_TOT_CYC
			</sampling>
			</set>
			<set enabled="yes" domain="user" changeat-globalops="5">
			PAPI_TOT_INS,PAPI_FP_INS,PAPI_TOT_CYC
			</set>
		</cpu>
		<network enabled="no"/>
		<resource-usage enabled="no"/>
		<memory-usage enabled="no"/>
	</counters>
```

```
<storage enabled="no">
        <trace-prefix enabled="yes">TRACE</trace-prefix>
        <size enabled="no">5</size>
        <temporal-directory enabled="yes">/tmp</temporal-directory>
        <final-directory enabled="yes">/home/clusty/MP/paral_MPI/tests/paral_MPIO</final-directory>
        <gather-mpits enabled="no"/>
</storage>
<buffer enabled="yes">
        <size enabled="yes">150000</size>
        <circular enabled="no"/>
</buffer>
<trace-control enabled="yes">
        <file enabled="no" frequency="5M">/gpfs/scratch/bsc41/bsc41273/control</file>
        <global-ops enabled="no"/>
        <remote-control enabled="no">
                <signal enabled="no" which="USR1"/>
        </remote-control>
</trace-control>
<others enabled="yes">
        <minimum-time enabled="no">10M</minimum-time>
</others>
<bursts enabled="no">
        <threshold enabled="yes">500u</threshold>
        <mpi-statistics enabled="yes"/>
        <pacx-statistics enabled="yes"/>
</bursts>
<cell enabled="no">
        <spu-file-size enabled="yes">5</spu-file-size>
        <spu-buffer-size enabled="yes">64</spu-buffer-size>
        <spu-dma-channel enabled="no">2</spu-dma-channel>
</cell>
</trace>
```

Fitxer de configuració per tracejar amb extrae

Vegem que volen dir els tags més importants que apareixen el fitxer de configuració d'exemple:

- trace enables="yes": indica que volem generar fitxers de traces.
- home= especifica on està instal·lat el paquet d'instrumentació, sol ser el mateix que la variable MPITRACE_HOME.
- initial-mode=detail :dona informació detallada sobre la traça.
- type=paraver: especifica que els fitxers intermitjos seran usats per fer traces paraver. També es podrien especificar per a dimemas.
- mpi enabled="yes", counters enabled="yes": estem especificant que volem recolectar informació sobre mpi i que volem saber sobre el rendiment de les crides MPI.
- openmp enables="yes": especifiquem que volem recolectar informació sobre openMP, que no volem recolectar informació sobre els bloquejos de variables però, si que volem informació sobre les rutines.
- counters enabled="yes": indiquem que volem usar els comptadors hardware de rendiment. Podem crear un node <set> per cada configuració que volguem usar,

només una configuració serà usada en un daterminat moment en una tasca especifica. Cada set conté la llista dels contadors de rendiment que volem usar; si estem usant PAPI els contadors els hem d'especificar amb els noms canònics o bé amb el seu codi hexadecimal. Una altre paràmetre important a especificar és el domain. Si és kernel indica que només compti els events quan l'aplicació està corrent en mode kernel, si es user quan l'apliació està en mode espai d'usuari i si és all conta els events independentment del mode en que estigui corrent l'apliació. Es poden espeficiar contadors diferents per a cada mode d'execució.

Anem ara a enumerar les pases que cal seguir per fer traces des del principi:

- Definir les variables d'entorn per interceptar les crides.
  - export MPITRACE_HOME=/usr/local/
  - export MPTRACE_CONFIG_FILE=./mpitrace.xml
  - export LD_LIBRARY_PATH=/usr/local/lib/
  - export LIBXML2_HOME=/usr/
  - export LD_PRELOAD=libmpitrace.so

- Definir les variables d'entron d'OpenMP si el nostre codi està paral·lelitzat amb OpenMP.
  - export OMP_NUM_THREADS=2

- Tenir el fitxer de configuració de les opcions de traça .xml en el directori on s'executarà el programa.
- Aixecar el clúster.
  - mpdboot –n 2 –f /home/clusty/mpd2.hosts

- Correr el programa que volem monitoritzar amb la commanda següent:
  - mpiexec –n 2 ./hytest.c

- Un cop tenim el fitxers intermedis .mpit, concretament un per procès MPI. Els hem d'unir amb la comanda mpi2prv per generar la traça final per al Paraver. On el exemple de sota output.prv és el nom de la traça que farem servir en el Paraver i TRACE.mpits és el nom del fitxer on es troven els paths dels fitxers .mpit, per defecte configurat en el ftixer de configuració .xml:
  - ${MPITRACE_HOME}/bin/mpi2prv –f TRACE.mpits –o output.prv

13.3.1 Instal·lació Dimemas

Dimemas és una eina que ens permet fer una anàlisi de quin és el comportament que té un programa en una màquina, quin rendiment té sobre cada component de la màquina i quin comportament podria tenir en una altre màquina, [10], [20*].

- Anem a http://www.bsc.es/plantillaC.php?cat_id=625
- Descarreguem el paquet Dimemas 3.0 Linux-x86 (32 bits)
- Instal·lem Java en la seva versió més actual
- Instal·lem el següent paquet per a una llibreria
    - apt-get install openjdk-6-jre
    - La necessitat d'aquest paquet es pot deduir amb apt-file search /urr/bin/jvm/..../libmawt.so
- export DIMEMAS_HOME=/home/clusty/pathfinsalacarpeta/dimemas-3.00.i686
- ./dimemas-gui.sh

13.4 Taules de conversió principals

Desenvolupar en el àmbit de la computació d'alt rendiment requereix mesurar tres factors clau; la potència consumida, les dades a computar i auxiliars i les operacions per segon sobre les dades, [52*].

13.4.1 Potència

Submúltiples			Múltiples		
Valor en Watts	Símbol	Nom	Valor en Watts	Símbol	Nom
10^–1 W	dW	Deciwatt	10^1 W	daW	Decawatt
10^–2 W	cW	Centiwatt	10v2 W	hW	Hectowatt
10^–3 W	mW	Milliwatt	10^3 W	kW	Kilowatt
10^–6 W	µW	Microwatt	10^6 W	MW	Megawatt
10^–9 W	nW	Nanowatt	10^9 W	GW	Gigawatt
10^–12 W	pW	Picowatt	10^12 W	TW	Terawatt

Taula 13.1: Taula de conversió de potència

13.4.2 Dades

Nom	Abreviatura	Nombre de bytes
byte	B	2^0 = 1
Kilobyte	k	2^10 = 1024
megabyte	M	2^20 = 1 048 576
Gigabyte	G	2^30 = 1 073 741 824
Terabyte	T	2^40 = 1 099 511 627 776
Petabyte	P	2^50 = 1 125 899 906 842 624
Exabyte	E	2^60 = 1 152 921 504 606 846 976
Zettabyte	Z	2^70 = 1 180 591 620 717 411 303 424
Yottabyte	Y	2^80 = 1 208 925 819 614 629 174 706 176

Taula 13.2: Taula de conversió de dades

13.4.3 Operacions per segon

Nom	Nombre de flops
kiloflops	10^3
megaflops	10^6
gigaflops	10^9
teraflops	10^12
petaflops	10^15
exaflops	10^18
zettaflops	10^21

Taula 13.3: Taula de conversió d'operacions per segon

13.5 Amples de banda de components

13.5.1 LAN

Nom	Mbps	MBps
Gigabit Ethernet (1000BASE-X)	1,000 Mbit/s	125 MB/s
Myrinet 2000	2,000 Mbit/s	250 MB/s
Infiniband SDR 1X[24]	2,000 Mbit/s	200 MB/s
Quadrics QsNetI	3,600 Mbit/s	450 MB/s
Infiniband DDR 1X[24]	4,000 Mbit/s	400 MB/s
Infiniband QDR 1X[24]	8,000 Mbit/s	800 MB/s
Infiniband SDR 4X[24]	8,000 Mbit/s	800 MB/s
Quadrics QsNetII	8,000 Mbit/s	1,000 MB/s
10 Gigabit Ethernet (10GBASE-X)	10,000 Mbit/s	1,250 MB/s
Myri 10G	10,000 Mbit/s	1,250 MB/s
Infiniband DDR 4X[24]	16,000 Mbit/s	1,600 MB/s
Scalable Coherent Interface (SCI) Dual Channel SCI, x8 PCIe	20,000 Mbit/s	2,500 MB/s
Infiniband SDR 12X[24]	24,000 Mbit/s	2,400 MB/s
Infiniband QDR 4X[24]	32,000 Mbit/s	3,200 MB/s
40 Gigabit Ethernet (40GBASE-X)	40,000 Mbit/s	5,000 MB/s
Infiniband DDR 12X[24]	48,000 Mbit/s	4,800 MB/s
Infiniband QDR 12X[24]	96,000 Mbit/s	9,600 MB/s
100 Gigabit Ethernet (100GBASE-X)	100,000 Mbit/s	12,500 MB/s

Taula 13.4: Taula d'amples de banda de xarxes LAN

13.5.2 Busos interns

Nom	Mbps	MBps
InfiniBand single 4X[24]	8,000 Mbit/s	1,000 MB/s
UPA	15,360 Mbit/s	1,920 MB/s
PCI Express 1.0 (x8 link)[39]	16,000 Mbit/s	2,000 MB/s
AGP 8x	17,066 Mbit/s	2,133 MB/s
PCI-X DDR	17,066 Mbit/s	2,133 MB/s
HyperTransport (800 MHz, 16-pair)	25,600 Mbit/s	3,200 MB/s
HyperTransport (1 GHz, 16-pair)	32,000 Mbit/s	4,000 MB/s
AGP 8x 64-bit	34,133 Mbit/s	4,266 MB/s
PCI Express (x32 link)[39]	64,000 Mbit/s	8,000 MB/s
PCI Express 3.0 (x16 link)[41]	102,400 Mbit/s	12,800 MB/s
PCI Express 2.0 (x32 link)[41]	128,000 Mbit/s	16,000 MB/s
QPI (4.80GT/s, 2.40 GHz)	153,600 Mbit/s	19,200 MB/s
HyperTransport (2.8 GHz, 32-pair)	179,200 Mbit/s	22,400 MB/s
QPI (5.86GT/s, 2.93 GHz)	187,520 Mbit/s	23,440 MB/s
PCI Express 3.0 (x32 link)[40]	204,800 Mbit/s	25,600 MB/s
QPI (6.40GT/s, 3.20 GHz)	204,800 Mbit/s	25,600 MB/s
HyperTransport 3.1 (3.2 GHz, 32-pair)	409,600 Mbit/s	51,200 MB/s

Taula 13.5: Taula d'amples de banda de busos interns

13.5.3 Emmagatzematge

Nom	Mbps	MBps
Ultra-320 SCSI (Ultra4 SCSI) (16 bits/80 MHz DDR)	2,560 Mbit/s	320 MB/s
Fibre Channel 4GFC (4.25 GHz)[43]	3,400 Mbit/s	425 MB/s
Serial ATA 3 (SATA-600)[44]	4,800 Mbit/s	600 MB/s
Ultra-640 SCSI (16 bits/160 MHz DDR)	5,120 Mbit/s	640 MB/s
Fibre Channel 8GFC (8.50 GHz)[43]	6,800 Mbit/s	850 MB/s
AoE over 10GbE, per path	10,000 Mbit/s	1,250 MB/s
FCoE over 10GbE	10,000 Mbit/s	1,250 MB/s
iSCSI over InfiniBand 4x	40,000 Mbit/s	5,000 MB/s
iSCSI over 100G Ethernet (hypothetical)	100,000 Mbit/s	12,500 MB/s

Taula 13.6: Taula d'amples de banda de busos interns

13.5.4 Memòries RAM

Nom	Rate bits/s	Rate bytes/s	Rate (MHZ)	DDR #
PC2-5400 DDR2-SDRAM (dual channel 128-bit)	85.3 Gbit/s	10.7 GB/s	667 MHz	DDR2-667
PC2-6400 DDR2-SDRAM (dual channel 128-bit)	102.4 Gbit/s	12.8 GB/s	800 MHz	DDR2-800
PC2-8000 DDR2-SDRAM (dual channel 128-bit)	128.0 Gbit/s	16.0 GB/s	1,000 MHz	DDR2-1000
PC2-8500 DDR2-SDRAM (dual channel 128-bit)	136.0 Gbit/s	17 GB/s	1,066 MHz	DDR2-1066
PC3-17066 DDR3-SDRAM (dual channel 128-bit)	273.1 Gbit/s	34.1 GB/s	2,133 MHz	DDR3-2133
PC3-17066 DDR3-SDRAM (triple channel 192-bit)	409.6 Gbit/s	51.2 GB/s	2,133 MHz	DDR3-2133
PC3-17600 DDR3-SDRAM (dual channel 128-bit)	281.6 Gbit/s	35.2 GB/s	2,200 MHz	DDR3-2200
PC3-17600 DDR3-SDRAM (triple channel 192-bit)	422.4 Gbit/s	52.8 GB/s	2,200 MHz	DDR3-220

Taula 13.7: Taula d'amples de banda de memòries RAM

14 Glossari

14.1 Conceptes generals

Dynamic frequency scaling (DFS): Escalat de freqüència dinàmic. Serveix per variar la freqüència d'un processador o un bus i així adaptar el consum segons les necessitats.

Sistem on a chip (SoC): Sistema en un chip. Es tracta d'un sistema que es pot composar de processadors de propòsit específic processadors de so, processadors d'imatge, etc. imprès en un sol chip.

ECC (Error checking and correction): És una important pràctica per al manteniment i integritat de les dades a través de canals amb soroll i mitjans d'emmagatzemament poc fiables, per exemple aplicat a les memòries DRAM. És imprescindible si es vol computar sense errors. Les GPUs normals no en porten la qual cosa les diferencia de les GPGPUs.

Memory Management Unit (MMU): O també anomenat paged memory management unit, (PMMU). S'encarrega d'aconseguir la traducció de les adreces lògiques,(virtuals), a les físiques, (reals), protegir els accessos prohibits a memòria i controlar la memòria cau.

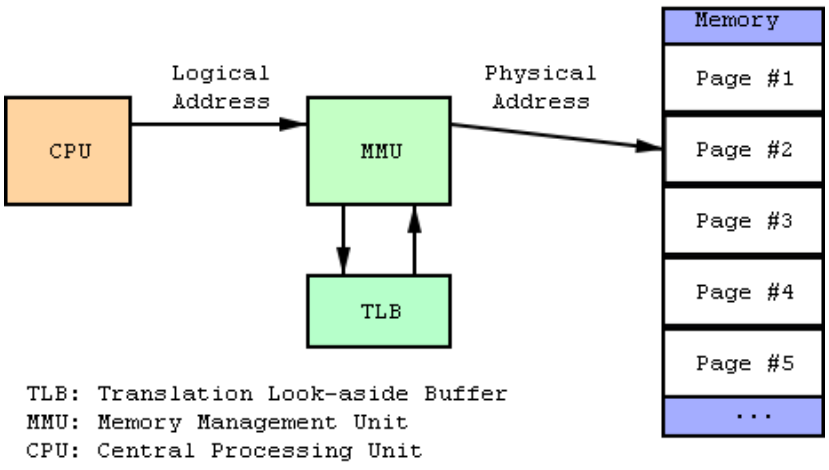

Figura 14.1: Esquema del funcionament de la traducció d'adreces virtuals a físiques

Quan la CPU vol accedir a una adreça de memòria lògica la MMU consulta primer el TLB, que és una memòria cau amb les ultimes traduccions de la taula de pagines usades, de adreces lògiques a adreces reals, si la traducció, (anomenada page table entry, PTE),no es troba a el TLB ha de consultar la taula de pagines del procés a memòria principal. En cas de no trobar-se la pagina a memòria principal es produeix una

fallada de pagina, on el sistema operatiu ha de fer la o les substitucions necessàries mitjançant certs algorismes que poden variar en funció del sistema operatiu. Un dels avantatges principals del MMU es que pot protegir a un procés d'accessos il·legals al seu espai d'adreces per part d'altres processos.

Aplication binary interface (ABI): Descriu la interfície de baix nivell entre l'aplicació o algun tipus de programa i el sistema operatiu o una altre aplicació.

Descriuen detalls com ara els tipus de dades, la seva mida, l'alineament, la convenció de crides a funcions, que controla com es passen els arguments i com es retornen els resultats; els nombres de crides a sistema i com una aplicació ha de fer les crides a sistema al sistema operatiu.

Fab: És la fàbrica on es fan les oblies i on s'imprimeixen amb els transistors, que com ja sabem s'incrementen en nombre segons la llei de Moore. Les fabs solen pertànyer a la companyia que ven el chip, com per exemple, AMD, Intel, Texas Instruments, o Freescale. Una fosa, (foundry), és una fab on els chips o les oblies són venudes a altres companyies, (third party), que venen el chip. Exemples són, Taiwan Semiconductor Manufacturing Company (TSMC), United Microelectronics Corporation (UMC) o Semiconductor Manufacturing International Corporation (SMIC).

Taiwan Semiconductor Manufacturing Company, Limited (TSMC): Fàbrica, (fab), de semiconductors independent més grossa del món, situada a Taiwan. Ofereix una amplia gama de oblies per imprimir els circuits. Les següents companyies en són clients: Applied Micro Circuits Corporation, Qualcomm, Altera, Broadcom, Conexant, Marvell, NVIDIA, i VIA. També Intel delega una part de la seva producció.

El 2006 va tenir un guanys de 3,63 bilions de dollars. Ha anunciat invertir 9,4 bilions de dòlars per fer una fabrica d'oblies de tecnologia de 40 i 20 manòmetres, esperada pel Març de 2012, s'espera que faci 100000 oblies al més i generi uns ingressos de 5 bilions per any.

Aquesta fàbrica és d'especial interès i prové a ARM i NVIDIA.

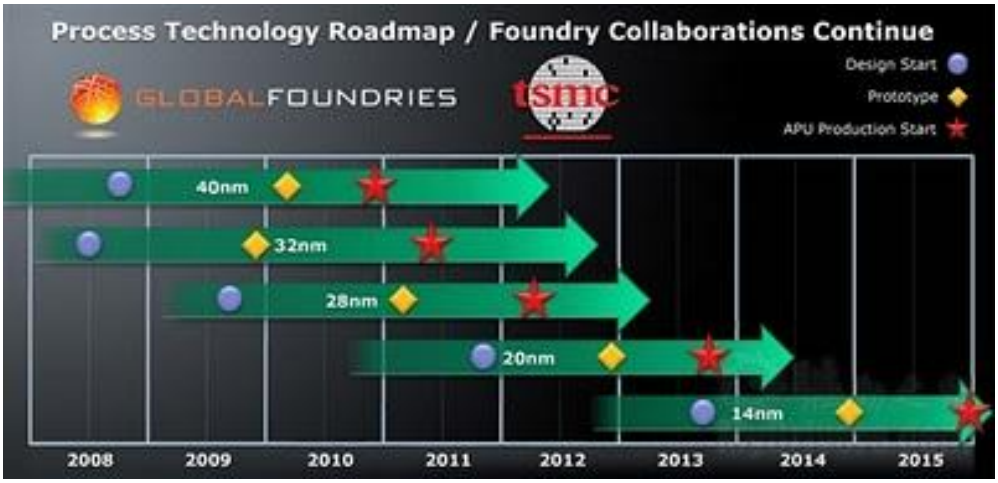

Figura 14.2: Projecció de les tecnologies de transistor pel futur

Tick-Tock: Model adoptat pel fabricant de xips Intel Coporation des del 2007. Tracta de seguir al canvi de mircroarquitectura amb un encongiment d'aquesta, gràcies al encongiment del transistor, (podem veure la seva projecció en la figura 14.2). Cada "tick" és un encongiment de microarquitectura vella i cada "tock" es un canvi de microarquitectura. Per cada any s'espera mínim un "tick" o un "tock", [34*], [35*], [36*].

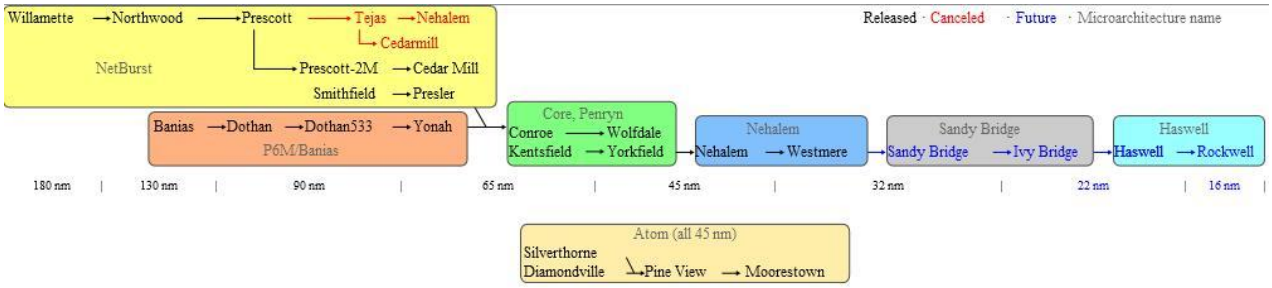

Figura 14.3: Microarquitectures i tecnologies de manòmetres usades de Intel

14.3 Clustering

Rsync: És una aplicació per sistemes de tipus Unix que ofereix transmissió eficient de dades incrementals, compressió de dades i xifrats. Mitjançant un tècnica de delta encoding, permet sincronitzar arxius i directoris entres dues màquines d'una xarxa o entre dues ubicacions d'una mateixa màquina, minimitzant el volum de dades transferides. Una característica important de rsync no trobada en la majoria de programes o protocols es que la copia pren lloc amb una sola transmissió en cada direcció . Rsync pot copiar o mostrar directoris continguts i copia d'arxius, opcionalment usant compressió.

En modalitat "Daemon" servidor, rsync escolta per defecte en el port TCP 873, servint arxius den protocol natiu rsync o via un terminal remot com RSH o SSH. EN l'últim cas, l'executable el client rsync ha de ser instal·lat en el host local i remot.

Llançat sota llicència GNU General Public Licence, rsync es software lliure, [21*].

Network Information Service NIS: Sistema d'informació en xarxa, és un protocol de servies de directoris client-servidor desenvolupat per Sun Microssitems per a enviar dades de configuració en sistemes distribuïts tals com noms d'usuaris i hosts entre computadores sobre una xarxa.

NIS proporciona prestacions d'accés a bases de dades genèriques que es poden usar per a distribuir, per exemple, la informació continguda en els fitxers /etc/passwd i /etc/groups a tots el nodes de la seva xarxa. Això fa que la xarxa sembli un sistema individual, amb totes les mateixes contes en tots els nodes. De manera similar, es pot usar NIS per distribuir informació de noms de node continguda en /etc/hosts a totes les maquines d'una xarxa.

14.4 Tecnologies ARM

Advanced Microcontroller Bus Architecture (AMBA): És una família d'especificacions de protocol que descriuen una estratègia per inteconnectar. És el estàndard obert per a busos on-chip. És una especificació de bus on-chip que detalla una estratègia per interconnectar i gestionar els blocs funcionals que conformen el SoC, (System-on-Chip). Ajuda al desenvolupament de processadors amb una o més CPUs o processadors de senyals amb múltiples perifèrics. AMBA complementa una metodologia de disseny reusable definint un "backbone", (columna vertebral), comú per als mòduls del SoC.

Advanced eXtensible Interface (AXI): És un protocol de bus que suporta separar les fases de adreces/control de la de dades, transferències de bytes desalineats usant "byte strobes", transferències en rafega basant-se només amb la primera adreça, canals de lectura i escriptura separats per aconseguir DMA a baix cost, capacitat per suportar múltiples peticions al bus "outstanding", (al vol), completesa de transaccions fora d'ordre, fàcil adició de fases de registres per proporcionar data de tancament. També té extensions opcionals per proporcionar senyals de control per operació en mode de baix consum. AXI és una de les interfícies que es poden fer amb la família d'especificacions AMBA, [5].

Thumb instruction: Tecnologia d'instruccions d'ARM que tenen una longitud menor,(la meitat d'una paraula), i per tant aprofiten millor l'espai de memòria. Com a conseqüència han d'estar alineades a la longitud que tinguin.

TrustZone: És una tecnologia, que proporciona seguretat a plataformes d'alt rendiment com és ara l' ARM Cortex-A9. Suporta un ampli ventall d'aplicacions incloent pagament segur, gestió de drets digitals, (DRM), i serveis basats en web.

Una de les característiques més rellevants és que la tecnologia s'estén pel bus AMBA AXI i als blocs IP dels sistema específics. La qual cosa vol dir que es possible assegurar perifèrics com són ara un teclat o un monitor. Per més informació consultar el lloc web d'ARM.

14.5 Llibreries de còmput

BLAS: Basic linear àlgebra subprograms, és una API estàndard per fer operacions d'àlgebra lineal bàsica, com ara multiplicacions de vectors i matrius. És usada per altres paquets més grossos com el LINPACK i el LAPACK. Té tres nivells: el primer són operacions un vector, el segon són operacions un vector i una matriu, i el tercer són operacions entre dues matrius, [42].

BLACS: Basic Linear Algebra Communication Subprograms; és un interfície MPI per a àlgebra lineal implementada eficientment per a plataformes de memòria distribuïda.

Com que paral·lelitzar algorismes per a cada màquina és una tasca molt costosa, BLACS existeix per poder programar més fàcilment els algorismes d'àlgebra lineal.

PBLAS: També anomenat parallel BLAS; entre altres coses permet l'adreçament global de matrius distribuïdes.

LINPACK: És un benchmark molt usat en HPC. Principalment fa crides a la subrutina General DAXPY de la biblioteca BLAS que fa l'operació y(i) := y(i) + a * x(i). Amb nombres de coma flotant de dobles precisió. Descriu el rendiment del sistema per resoldre un problema de matrius generals denses Ax=b a tres nivells de mida i amb oportunitat a optimització. Les matrius són generades aleatòriament però, forçant als nombres perquè puguin executar-se amb pivots parcials amb eliminació Gaussiana. Està escrit en Fortran així que els recorreguts de les matrius dels codis es fan per columnes.

LAPACK: És el benchmark que ha reemplaçat àmpliament al Linpack, inclou factoritzacions de matrius com ara LU, QR, Cholesky i descomposició Schur. Està escrit en Fortran.

ScaLAPACK: És la versió del Lapack per a sistemes de memòria distribuïda com ara clústers, [49*], [46*].

Esquema del conjunt:

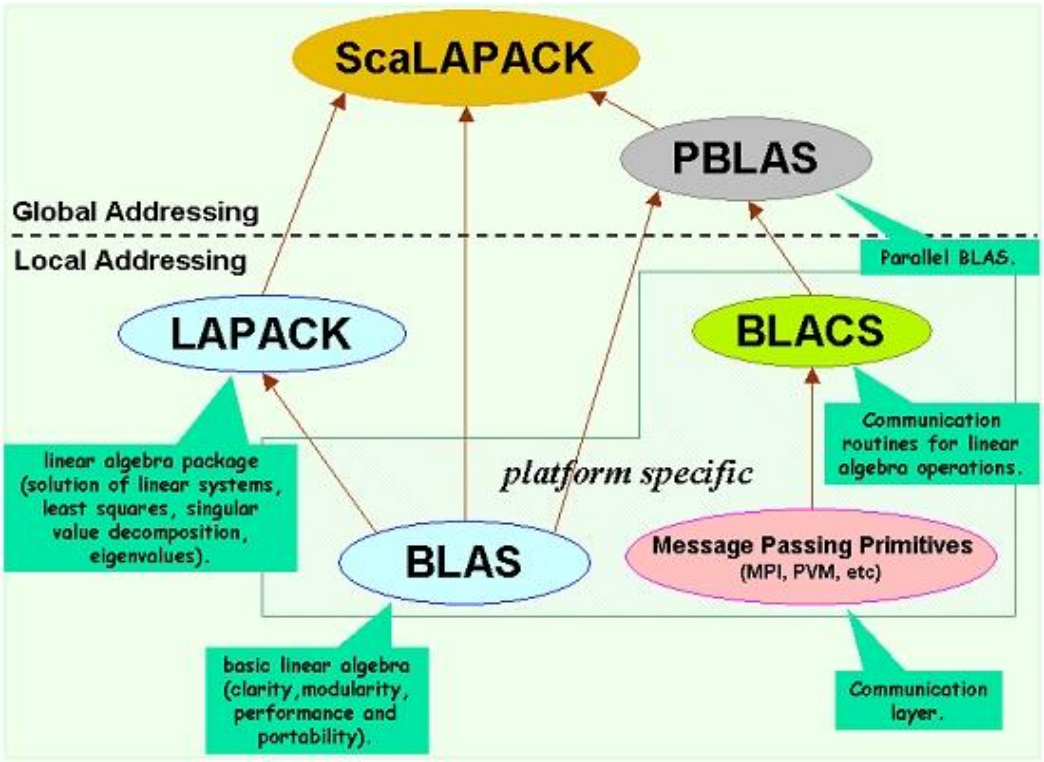

Figura 14.4: Esquema de dependències de les llibreries de còmput

14.6 Xarxes d'interconnexió

Fast-Ethernet: És el nom d'una sèrie d'estàndards de lEEE de xarxes Ethernet de 100 Mbps. El nom Ethernet ve del concepte físic de ehter. El prefix fast es va posar per diferenciar-lo de l'original Ehternet de 10 Mps.

Gigabit Ethernet: També coneguda com GigaE, és una ampliació de l'estàndard Ethernet que assoleix una capacitat de transmissió de 1 gigabit per segon.

100 Gigabit Ethernet (XGbE o 10GbE): Defineix una versió de Ethernet amb una velocitat nominal de 10 Gbits/s, deu vegades més ràpid que gigabit Ethernet.

Myrinet: Xarxa d'interconnexió usada en clústers d'altes prestacions. Actualment la versió Myri-10G arriba als 10 Gbps amb latències de 3 microsegons, y pot interoperar amb 10 Gb Ethernet. Una de les seves característiques es que el processament de les comunicacions de la xarxa es fa a traves de xips integrats en les targes de xarxa de Myrinet, (Lanai xips), descarregant a la CPU de gran part del processament de les comunicacions.

En quant al middleware de comunicació, la immensa majoria està desenvolupat per Myricom. Destaquen les llibreries de baix nivell GM i MX. (encaminament estàtic o adaptatiu).

Infiniband: És un bus de comunicacions sèrie bidireccional d'alta velocitat, dissenyat tant per connexions internes com per externes. Les seves especificacions són desenvolupades per la Infiniband trade Association, (IBTA).

Infiniband es pot categoritzar segons el ritme en el que es poden fer les transferències: úniques, (SDR), dobles ,(DDR) i quadrubles, (QDR); i segons el nombre d'enllaços afegits que poden ser 1x, 4x o 12x.

Té un latència mínima de 6 microsegons i en la seva versió QDR 12x pot arribar a un ample de banda de 96 Gbps.

NUMAlink: És un sistema d'interconnexió de SGI per a usar en els seus sistemes de memòria compartida distribuïda (distributed shared memory), ccNUMA, (cache coherent NUMA). La versió NUMAlink 5 va ser introduïda el 2009 i usada per les màquines Altix UV series. NUMAlink 5 és capaç 15GB/s d'ample de banda de pic, a través de dos links unidireccionals de 7,5BG/s amb latències per sota dels 2 microsegons.

14.7 Profiling

PAPI: Performance Application Programming Interface; és un projecte que pretén estandarditzar i implementar una API portable i eficient, per accedir als comptadors hardware de rendiments de la majoria dels processadors moderns. Per poder usar-la cal, entre altres coses, suport del kernel del SO, [17], [48*], [51*].

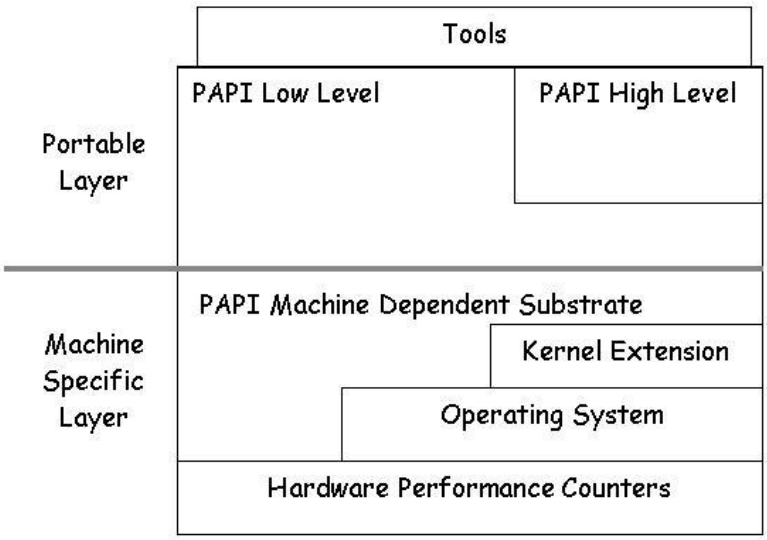

Figura 14.5: Esquema de la estructura funcionament de PAPI

Dyninst: És una llibreria multiplataforma de temps d'execució per aplicar sobre un codi. Permet interceptar les crides que desitgem per fer mesures de rendiment; per exemple per interceptar les crides a MPI_send() i saber el temps que representen d'una execució. Ha estat desenvolupada a les universitats de Wisconsin–Madison i Maryland, College Park. Ens pot ser molt útil, per exemple, quan no disposem d'accés als comptadors hardware del processador.

14.8 Emmagatzematge de dades

NFS (Network File System): És un protocol de sistema de fitxers en xarxa situat en la capa d'aplicació en la pila de protocols OSI, que permet a una computadora client accedir a fitxers a través de xarxa fàcilment, com si els dispositius físics d'emmagatzemament ,(normalment discs durs), estiguessin directament connectats a l'ordinador. NFS funciona sobre el protocol RPC. S'especifica al RFC 1094, RFC 1813 i RFC 3530 (que deixa obsolet el RFC 3010), [31*], [32*].

DAS (Direct Attached Storage): Consisteix en connectar el dispositiu d'emmagatzematge directament al servidor o estació de treball, és a dir, físicament connectat al dispositiu que fa ús de ell. Les aplicacions i programes d'usuaris fan les seves peticions de dades al sistema de fitxers directament. En una DAS, l'emmagatzematge és local respecte al sistema de fitxers.

SAN (Storage Area Network): Arquitectura de xarxa per a connectar dispositius d'emmagatzematge informàtic (com matrius de discs, biblioteques de cintes i dispositius òptics) a servidors, de manera que per al sistema operatiu els dispositius apareixen com connectats localment Les aplicacions i programes d'usuaris fan les seues peticions de dades al sistema de fitxers directament. En una SAN, l'emmagatzematge és remot. Les SAN són usades més típicament en les grans empreses ja encara que tenen més rendiment són més cares que les NAS. NAS es pot considera una extensió de DAS, on DAS hi ha un enllaç punt a punt entre el servidor i el servidor i el seu emmagatzemament, una SAN permet a varis servidors accedir a varis dispositius d'emmagatzemament en una xarxa compartida.

NAS (Network Attached Storage): És una tecnologia d'emmagatzemament dedicada a compartir la capacitat d'emmagatzematge d'un computador, (servidor; amb ordinadors personals o servidors clients a traves d'una xarxa), normalment TCP/IP, fent us d'un sistema operatiu optimitzat per donar accés amb els protocols CIFS, NFS, FTP o TFTP. NAS treballa amb transferències de fitxers complets, amb la qual cosa el client demana tot el fitxer al servidor podent el client treballar localment amb tot el fitxer. Molts sistemes NAS fan ús de disposicions RAID dels discs per incrementar la capacitat de transferència així com redundància.

DAS VS NAS VS SAN: Les línies discontinues vermelles ens indiquen que els components encerclats estan dins d'una mateixa màquina.

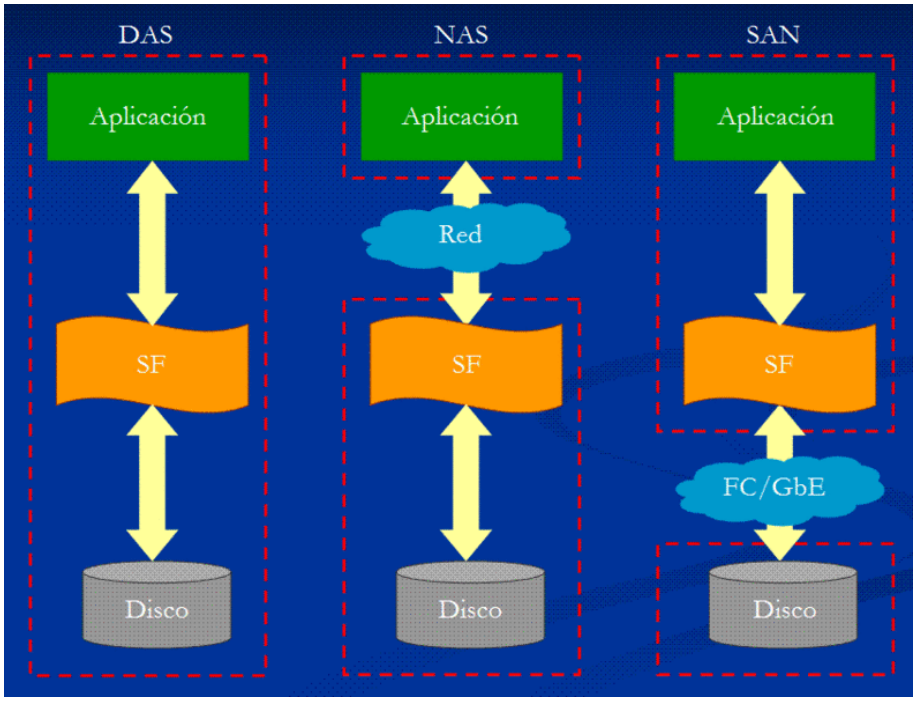

Figura 14.6: Esquema de les diferents tecnologies d'emmagatzemament a disc

- DAS: L'aplicació, el sistema de fitxers i el disc estan en una mateixa màquina, pel que no cal xarxa per al sistema d'emmagatzematge.
- NAS: L'aplicació fa peticions a la xarxa directament, la màquina remota té el sistema de fitxers i el disc físic. En una NAS les peticions són més aviat una sol·licitud de porció d'arxiu més que un bloc de disc. NAS té menys rendiment i fiabilitat que DAS i SAN.
- SAN: En una San l'aplicació i el sistema de fitxers estan a la màquina local, per tant l'aplicació accedeix al disc físic per la xarxa de manera transparent encara que estigui en una altre màquina. Les SAN usen xarxa de fibra òptica o més recentment iSCSI que és barat i usa TCP/IP per fer les transferències de dades.

15 Bibliografia

[1] Nvidia Corporation, Tegra_Multiprocessor_Architecture_white_paper_Final_v1.1, www.nvidia.com.

[2] Nvidia Corporation, tegra_250_hw_setup, www.nvidia.com.

[3] Nvidia Corporation, linux_for_tegra_quickstart_20100709, www.nvidia.com.

[4] ARM Corporation, ARMCortexA-9Processors, www.arm.com.

[5] ARM Corporation, AMBA AXI Protocol, specification, www.arm.com.

[6] Tatsuya Kobayashi, Transparències del fòrum ARM de Japó, www.jp.arm.com.

[7] John Goodacre, “The effect and technique of System Coherence in ARM Multicore Technology”, www.mpsoc-forum.org.

[8] ARM Corporation, Cortex-A9 Floating-Point Unit Revision:r2p2 Technical Reference Manual, www.arm.com.

[9] Roger Espasa, Transparències d’ACA de la FIB d’execució fora d’ordre, www.fib.upc.edu.

[10] European Center for Parallelism of Barcelona, Transparències de Dimemeas del BSC, www.bsc.es.

[11] Steve Chen, Neuroscience, Bio-system science, Thrid_Brain and Bio-Supercomputing, www.codata.org.

[12] Xavier Abellán Écija, Adaptació i desenvolupament d’un sistema de mesures de rendiment per a Marenostrum, upcommons.upc.edu.

[13] David Vicente Dorca, Paral·lelització híbrida (MPI+STARSs) d’aplicacions d’alt rendiment, upcommons.upc.edu.

[14] BSC,Company MPITrace User Manual, inclòs en el paquet de binaris Extrae 2.0, www.bsc.es.

[15] Harald Servat Gelabert & Germán Llort Sánchez, MPItrace User guide manual, www.bsc.es.

[16] CEPBA, MPItrace tool version 1.1 Instrumentation of generic MPI applications User’s guide, www.bsc.es.

[17] Shirley Moore, Code Profiling Tools Shirley Moore Innovative Computing laboratory University of Tennessee, www.cs.utk.edu.

[18] Oak Ridge, National Laboratory, Exploring science at petascale , http://www.nccs.gov.

[20] Richard Walsh Steve Conway, Earl C. Joseph, Ph.D. Jie Wu, Technology cell IDC report on PowerXCell.

[21] Akinwummi (Akin) Odenivi, MPI one sided one two sided communication. Computer science colloquium, www.usd.edu.

16 Enllaços web

[1*] http://developer.nvidia.com/tegra

[2*] http://www.arm.com/products/processors/

[3*] http://www.cs.virginia.edu/stream/#StandardResults

[4*] http://www.tux.org/~mayer/linux/bmark.html

[5*] http://www.toradex.com/files/media/pdf/Colibri%20Modules_20100713-web.pdf

[6*] http://git.mansr.com/?p=linux-omap;a=commitdiff;h=5170038

[7*] http://ark.intel.com/Product.aspx?id=27501&processor=&spec-codes=SL6WE,SL6WG,SL792

[8*] http://ark.intel.com/Product.aspx?id=28001&processor=9030&spec-codes=SL9DH,SL9P9

[9*] http://www.keil.com/support/man/docs/armcc/armcc_cjaidcjb.htm

[10*] http://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html

[11*] http://www.arm.com/products/tools/software-development-tools.php

[12*] http://en.wikipedia.org/wiki/ARM_architecture

[13*] http://en.wikipedia.org/wiki/Dhrystone

[14*] http://www.top500.org/

[15*] http://www.green500.org

[16*] http://products.amd.com/en-us/OpteronCPUResult.aspx?f1=Six-Cre+AMD+Opteron%E2%84%A2

[17*] http://en.wikipedia.org/wiki/Xeon

[18*] http://www.nvidia.com/object/product_tesla_C2050_C2070_us.html

[19*] http://www.delltechcenter.com

[20*] http://www.bsc.es/plantillaF.php?cat_id=52

[21*] http://es.wikipedia.org/wiki/Rsync

[22*] http://en.wikipedia.org/wiki/Supercomputer

[23*] http://www.tomechangosubanana.com/tesis/escrito-1-split/node1.html

[24*] http://www.estrellateyarde.org/discover/cluster-beowulf-mpi-en-linux

[25*] http://robotics.ee.uwa.edu.au/courses/adv-comp-arch/labs/mpi/mpi_at_home.html

[26*] http://www.hq.nasa.gov/hpcc/petaflops/app.sw.stevens.html

[27*] http://nanos.ac.upc.edu/publications

[28*] http://www.bsc.es/plantillaG.php?cat_id=328

[29*] http://code.google.com/p/math-neon/source/browse/trunk/math_mat4.c

[30*] http://tldp.org/HOWTO/html_single/Beowulf-HOWTO/

[31*] http://www.linuxparatodos.net/portal/staticpages/index.php?page=12-como-nfs
[32*] http://insidehpc.com/2009/08/31/multi-pflops-supercomputer-roadmap/
[33*] http://insidehpc.com/2010/06/29/arm-funded-by-eu-to-build-low-power-datacenter/
[34*] http://en.wikipedia.org/wiki/High-k_dielectric
[35*] http://techresearch.intel.com/articles/Tera-Scale/1449.htm
[36*] http://techresearch.intel.com/articles/terascalepop.html
[37*] http://nfs.sourceforge.net/nfs-howto/ar01s03.html
[38*] http://www.debianhelp.co.uk/nfs.htm
[39*] http://www.cas.mcmaster.ca/~qiao/courses/cs748/mpi/mpicomp.html
[40*] https://www.cs.indiana.edu/Facilities/FAQ/Security/openssh.html
[41*] http://www.cyberciti.biz/faq/how-do-i-start-and-stop-nfs-service/
[42*] http://en.wikipedia.org/wiki/General_Matrix_Multiply
[43*] http://wiki.freepascal.org/MPICH
[44*] http://mpd.sourceforge.net/doc/mpd11.html
[45*] http://www.ubuntu-es.org/node/18765
[46*] http://mate.uprh.edu/~pnm/notas4061/cap3/sislin1.html
[47*] http://wiki.freepascal.org/MPICH
[48*] http://icl.cs.utk.edu/PAPI/forum
[49*] http://www.netlib.org/scalapack/
[50*] http://www.dyninst.org/
[51*] http://icl.cs.utk.edu/papi/forum/
[52*] http://en.wikipedia.org/wiki/List_of_device_bandwidths
[53*] http://en.wikipedia.org/wiki/Kerrighed
[54*] http://www.scl.ameslab.gov/Projects/mpi_introduction/para_pingpong.html
[55*] http://www.scl.ameslab.gov/Projects/NetPIPE/
[56*] http://www.scl.ameslab.gov/Projects/old/ClusterCookbook/nprun.html
[57*] http://www.duke.edu/~hpgavin/gnuplot.html
[58*] http://www.mitchellwebdesign.com/arm/lecture5/lecture5-5-2.html
[59*] http://htmlcoderhelper.com/arm-to-c-calling-convention-registers-to-save/